UNIVERSIDADE FEDERAL DE SANTA CATARINA

CENTRO TECNOLÓGICO - CTC

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA - INE

CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

RONAN ROMEU KNOB

Análise e *benchmarking* das soluções NewSQL CockroachDB, MemSQL, NuoDB e VoltDB

Florianópolis

2018

RONAN ROMEU KNOB

Análise e *benchmarking* das soluções NewSQL CockroachDB, MemSQL, NuoDB e VoltDB

Monografia apresentada ao curso de graduação em Sistemas de informação, como parte dos requisitos necessários à obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Ronaldo dos Santos Mello
Coorientador: Geomar André Schreiner

Florianópolis
2018

Agradecimentos

Encerro esta etapa da graduação, tão importante na minha vida, e seria injusto não deixar agradecidas pessoas a quem serei eternamente grato, e que foram inseparáveis do meu sucesso nesta fase.

Agradeço á minha mãe e meu pai, bases fundamentais da pessoa que sou hoje, e que são a razão do sucesso nesta e em todas as fases da minha vida, que sempre deram atenção especial á educação dos seus filhos, e sempre batalharam para dar á mim e meus irmaõs a melhor condição possível.

Agradeço especialmente a meus amigos Giancarlo Souza e Sabrina Schütz, que têm minha eterna amizade e carinho, me ajudando a atravessar toda a graduação me dando a força que precisei para estar concluindo esta fase hoje.

Agradeço também a minha namorada Maria Clara Reschke, que neste ano de construção deste trabalho me apoiou com carinho e amor em todos os momentos, me ouvindo e consolando nos momentos difíceis e convivendo com minha ausência em alguns momentos, e que hoje compartilha comigo da felicidade por alcançar esse tão esperado momento.

Deixo também meus agradecimentos a todos os meus amigos, pessoas maravilhosas que tenho a oportunidade de ter ao meu lado. Apesar de não citar o nome de todos aqui, quero que saibam que tem minha eterna gratidão e amizade, por me apoiarem nessa fase, compreendendo momentos meus momentos de ausência, e me fortalecendo todos os dias com seu companheirismo.

Gostaria de deixar um agradecimento especial ao meu co-orientador Geomar André Schreiner, que me orientou em todos os momentos da construção desse trabalho sem medir esforços para tirar todas as minhas dúvidas; ao professor Ronaldo dos Santos Mello que acreditou em mim em todos os momentos e prestou toda a ajuda que eu necessitava. Por fim, agradeço a esta universidade, seu corpo docente, direção e administração, que forneceram a estrutura necessária para que tudo isso fosse possível.

Resumo

Os avanços em tecnologias Web e na proliferação de dispositivos móveis e conectados á internet, criaram no âmbito da informática uma necessidade de armazenamento de grandes quantidades de dados. O primeiro paradigma a tentar organizar esses dados foi o relacional. O aumento exponencial nos volumes de dados armazenados com o passar dos anos, e, ao mesmo tempo, a necessidade de recuperar informações com agilidade, fizeram emergir novos paradigmas em bancos de dados como é o caso do NoSQL (*Not Only SQL*). Tais paradigmas foram importantes, mas sacrificavam características dos bancos tradicionais, como a normalização e integridade. Um cenário comum em empresas é utilizar o mesmo banco de dados para processamento OLTP e OLAP. Isso quer dizer que não pode haver falta de integridade, mas, ao mesmo tempo, as transações operacionais não podem sofrer com o processo. Neste trabalho, o paradigma estudado é o NewSQL, criado com o intuito de integrar os benefícios dos pardigmas anteriores, oferecendo entre suas características transações que respeitam as propriedades ACID, nativamente distribuídas para oferecer velocidade e sem concorrência bloqueante.

Para esta análise, empregou-se a técnica de benchmark, utilizando de benchmarks de domínio específico, através de um *framework* chamado OLTP-Bench. Estes benchmarks envolvem um contexto de aplicação, o que demonstra de forma mais próxima da realidade o comportamento em um ambiente real. Foram escolhidos três benchmarks que apresentam cargas de diferentes tipos e complexidades, de modo a cobrir a análise do processamento das cargas por parte dos produtos.

Este trabalho comparou quatro produtos que empregam o paradigma NewSQL, através de um processo de *benchmarking*, através de três testes que empregam cargas de diferentes tipos e complexidades, de modo a cobrir a análise do processamento das cargas por parte dos produtos. A análise pode contribuir como referência para futuros usos da tecnologia.

Sobre os resultados obtidos das análises verificou-se que geralmente, o produto MemSQL se manteve á frente nas características observadas, obtendo alta taxa de *throughput*, e baixa latência nos contextos analisados. Também foi o único produto que conseguiu executar as cargas do teste TPC-H mostrando-se mais flexível que os demais produtos. Os produtos VoltDB e NuoDB se comportaram de maneira semelhante na maioria dos contextos analisados, mostrando também uma boa execução das cargas analisadas, porém não terminaram a execução dos testes do benchmark TPC-H assim como o produto CockroachDB, prejudicando a análise sobre o processamento OLAP por parte destes produtos.

Palavras chave: Bancos de dados, NewSQL, SGBD, benchmark

Abstract

Advances in Web technologies and the proliferation of mobile and internet-connected devices have created in the area of computing a need to store large amounts of data. The first paradigm to try to organize this data was the relational one. The exponential increase in the volumes of data stored over the years, and at the same time the need to retrieve information with agility, has given rise to new paradigms in databases such as NoSQL (Not Only SQL). Such paradigms were important, but they sacrificed characteristics of traditional banks, such as normalization and integrity. A common scenario in enterprises is to use the same database for OLTP and OLAP processing. This means that there can be no lack of integrity, but at the same time operational transactions can not suffer from the process. In this work, the paradigm studied is NewSQL, created with the intention of integrating the benefits of previous pardigms, offering among its characteristics transactions that respect ACID properties, natively distributed to offer speed and without blocking competition.

For this analysis, the benchmark technique was used, using specific domain benchmarks, through a framework called OLTP-Bench. These benchmarks involve an application context, which demonstrates behavior closer to reality in a real-world environment. Three benchmarks were chosen that present loads of different types and complexities, in order to cover the analysis of the processing of loads by the products.

This work compared four products that use the NewSQL paradigm, through a benchmarking process, using three tests that employ loads of different types and complexities, in order to cover the analysis of the processing of loads by the products. The analysis can contribute as a reference for future uses of technology.

The benchmark results show that in general the product MemSQL was ahead in the observed characteristics, obtaining high throughput rate and low latency. Also, it was also the only product that was able to perform the TPC-H test loads showing more flexibility than other products. The VoltDB and NuoDB products have similar behavior in most of the analyzed contexts, also showing a good execution of the analyzed loads, but can't finish the execution of the tests of the TPC-H benchmark as well as the product CockroachDB, damaging the analysis on the OLAP processing by part products.

Keywords: Databases, NewSQL, SGBDs, benchmark

Lista de ilustrações

Lista de gráficos

Lista de tabelas

Lista de abreviaturas e siglas

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BD	Banco de Dados
BI	Business Intelligence
CPU	Central Processing Unit - Unidade Central de Processamento
DB	Database - Banco de dados
DBMS	Database Management System
ERP	Enterprise Resource Planning
GB	Gigabyte
HTAP	Hybrid Transactional/Analytical processing
I/O	Input / Output
IP	Internet Protocol
JDBC	Java Database Connectivity
KV	Key-value
LTS	Long Time Suport - Suporte de longo prazo
ODBC	Open Database Connectivity
OLAP	Online Analytical Processing
OLTP	Online Transactional Processing
RAM	Random Access Memory - Memoria de Acesso Aleatorio
REST	Representational State Transfer
SF	Scale Factor - Fator de escala
SGBD	Sistema de Gerenciamento de Banco de Dados
SO	Sistema Operacional
SQL	Structured Query Language

SSH	Secure Shell
WAN	Wide Area Network

Sumário

1 Introdução

A necessidade de armazenar dados de forma ordenada e indexada hoje é indispensável para quase totalidade das organizações. Hoje, o valor da informação ultrapassa o valor dos próprios produtos da tecnologia(SHI et al., 2007). Considerando que o valor dessas informações é cada vez maior, é natural o esforço contínuo para coleta e armazenamento das massas de dados e o suporte computacional para tal deve evoluir de maneira a acompanhar esse ritmo.

O paradigma relacional foi o primeiro a se popularizar na armazenagem dessas massas de dados de forma estruturada. Os dados armazenados por este tipo de sistema respeitam as características ACID (*Atomicity, Consistency, Isolation, Durability*), que garantem atomicidade, consistência, isolamento e durabilidade aos dados. A arquitetura de informações do paradigma relacional se baseia em abstrações de entidades do mundo real, de forma normalizada e sistemática para facilitar o entendimento e a criação de pesquisas que conectam certas características de uma entidade com outras (PAVLO, 2016). Outro fator que ajudou a popularizar o paradigma relacional foi a linguagem de pesquisa para o paradigma, a SQL (*Structured Query Language*), devido á fácil abstração empregada. Apesar de intuitivo, o paradigma possui deficiências quanto á dimensionalidade. Quanto mais dados são armazenados, consultas ou outras operações que demandam muitos dados exigem alta capacidade computacional. Outro ponto é que estes sistemas possuem a característica de *lock* de tabelas (uma operação no banco pode travar uma tabela para manipulação até que a operação seja concluída), apresentando-se como um gargalo para sistemas que possuem como requisito a alta disponibilidade para um grande volume de dados (STONEBRAKER, 2011a).

Baseando-se nos desafios enfrentados pelo paradigma relacional tradicional, novos paradigmas de BDs começaram a ser desenvolvidos (SADALAGE; FOWLER, 2013). Em meados de 1998, com a evolução da quantidade de informação e busca por velocidade, surge o paradigma NoSQL (*Not Only SQL*). O paradigma emerge num momento em que sistemas de propósito único, assim como aplicações que precisam de acesso rápido aos dados se popularizam. O paradigma *NoSQL* tem como uma de suas principais características a escalabilidade horizontal pautada em uma arquitetura do tipo distribuída, geralmente com replicação e fragmentação dos dados em diferentes servidores (CATTELL, 2011).

BDs NoSQL não seguem o paradigma de dados relacionais e implementando novos modelos de dados, que suportam a representação de dados complexos (ABADI, 2009) . Estes modelos podem ser classificados em quatro categorias (SADALAGE; FOWLER, 2013): (i) chave-valor, onde um valor é armazenado com base em uma

chave e pesquisas no valor não são permitidas (e.g. Redis); (ii) orientado a colunas, que organizam seus dados utilizando famílias de colunas, colunas e valores (e.g. Cassandra); (iii) orientado a documentos, que armazena seus dados em documentos compostos por uma hierarquia de chaves e agregações por coleções de documentos (e.g. MongoDB); e (iv) orientado a grafos, que armazena todos os dados em um grafo e possui algoritmos de busca em grafos implementados (e.g. Neo4J).

BDs NoSQL geralmente não possuem esquema de dados e não possuem suporte a transações que respeitem a características ACID do paradigma relacional. Alguns possuem suporte ao conjunto de características BASE (*Basic avaliable, Soft-State e Eventually Consistent)*, onde há uma consistência eventual dos dados. Ainda, produtos baseados em NoSQL não oferecem nem suporte a BASE, ou seja, nenhum tipo de consistência.

Os sistemas relacionais geralmente trabalham com um escalonamento vertical de hardware, ou seja, quanto mais dados ou velocidade for necessária, é necessário investimento para incrementar o hardware do servidor centralizado onde está o banco de dados. A característica de *lock* de tabelas também impede que exista um processamento mais paralelizado das solicitações, o que influencia na velocidade. Algumas organizações, como instituições financeiras e sistemas de processamentos de pedidos necessitam que as transações sejam consistentes, mas, ao mesmo tempo precisam de velocidade para acompanhar a demanda atual, sendo um desafio para o paradigma relacional tradicional.

Apesar de o paradigma NoSQL tratar vários requisitos associados a Big data, ele também possui alguns pontos fracos. Como citado por Pavlo (PAVLO, 2016), algumas organizações como o próprio Google, chegaram á conclusão de que os desenvolvedores estavam gastando muito tempo escrevendo código para lidar com dados inconsistentes dos bancos não relacionais, e que usar transações era mais eficiente, pela abstração mais próxima do mundo real do paradigma relacional.

O paradigma NewSQL visa usufruir dos benefícios do paradigma relacional com a escalabilidade do paradigma NoSQL. São SGBDs modernos, que buscam prover o mesmo desempenho escalável do NoSQL, para cargas de trabalho OLTP (*Online Transactional Processing*) de leitura e escrita, enquanto mantém as garantias ACID para as transações(PAVLO, 2016, p. 46).

Em linhas gerais, os bancos classificados como NewSQL têm muitos dos requisitos para gerenciamento em ambientes de computação em nuvem (ou distribuída), que se popularizam cada vez mais, e também oferecem os benefícios dos padrões já conhecidos da SQL (GROLINGER et al., 2013). Uma das principais vantagens de usar um SGBD que é construído para uma execução distribuída é que suas partes são construídas com foco em distribuição de informações, diferente do cenário onde os produtos

são adaptados para tal funcionamento. Isso inclui possibilidades como otimizadores de consultas e protocolos de comunicação entre nodos (PAVLO, 2016).

A implementação de um banco de dados NewSQL deve considerar 2 importantes fatores: (i) Um controle de concorrência de esquema *lock-free*, e (ii) Uma arquitetura distribuída *shared-nothing* (STONEBRAKER, 2011b). O primeiro aspecto se aplica á questão da disponibilidade, a qual é afetada diretamente por *locks.* O segundo aspecto diz respeito a uma arquitetura de nodos, onde cada um é independente e auto-suficiente, e na arquitetura não há um ponto único de contenção.

Os bancos NewSQL compartilham algumas características globais como mencionado anteriormente. Porém, ainda são sensíveis ao contexto, ou seja, existem soluções diferentes para diferentes problemas. Então, para testar e avaliar esses produtos, são necessários *benchmarks* apropriados (JELLY; KERRIDGE; BATES, 1994). Os *benchmarks* são testes de simulação de aplicações reais, que envolvem cargas e estruturas (*schemas*) específicos (CURINO et al., 2012).

Um teste de benchmarking genérico resulta em resultados genéricos, devido à insensibilidade ao contexto. Neste trabalho, serão utilizados benchmarkings de domínio, que cobrem essa diversidade de usos. As cargas aplicadas categorizam aplicações típicas no domínio do problema. O desempenho dessa carga de trabalho em vários computadores nos dá uma estimativa de desempenho relativa desses sistemas em um domínio de problema (GRAY, 1993). Através do método podemos evidenciar os pontos fortes e negativos de cada solução, possivelmente evidenciando um melhor contexto de uso para cada solução.

Neste trabalho foram escolhidos sistemas NewSQL com versões de uso gratuito que permitissem os testes, escolhidas com base no ranking DB-Engines[1]. Este ranking leva em consideração aspectos como número de mencões em websites, interesse nas buscas (via *Google Trends*[2]), e outras fontes como a análise da pesquisa (*survey*) anual do site *Stack Overflow* no ano de 2017[3]. A pesquisa foi feita entre outubro e novembro de 2017. Também foi considerada a proposta dos sistemas em comparação com as características do paradigma NewSQL.

As soluções escolhidas com base nesses critérios foram os bancos de dados: VoltDB, NuoDB, MemSQL e CockroachDB. Foram utilizadas para o trabalho as versões gratuitas dos produtos, onde as limitações não interferem nos objetivos estipulados. Além disso, algumas soluções, como o CockroachDB e VolDB, são de código aberto.

[1] db-engines.com/
[2] https://trends.google.com.br/trends/
[3] https://insights.stackoverflow.com/survey/2017

1.1 Motivação

O paradigma NewSQL surgiu em meados de 2011 (STONEBRAKER, 2011b). É relativamente novo, e por isso não é apoiado por uma vasta documentação e material acadêmico de suporte, diferentemente do que acontece com o paradigma relacional tradicional, que possui sólida bibliografia (PAVLO, 2016) . O mesmo acontece com o paradigma NoSQL, porém em menor quantidade, visto que, enquanto o paradigma NoSQL tem aplicações mais específicas, o paradigma relacional continua sendo amplamente usado em razão de sistemas que lidam com o nível operacional das empresas no geral (sistemas de logística, sistemas ERP, etc), onde é necessário armazenar todas as atividades.

Existem hoje comparações entre produtos NoSQL (CHEVALIER et al., 2015), relacionais (STANCU-MARA; BAUMANN, 2008) e entre relacional e NoSQL (CATTELL, 2011)(FATIMA; WASNIK, 2016), mas a literatura carece de um material de mesmo porte para soluções NewSQL. O método de *benchmarking* escolhido para este trabalho pode auxiliar na escolha de um banco de dados, pois, ao aplicar uma carga de dados condizente ao que podemos encontrar em uma situação real, podemos avaliar como a aplicação se comportaria em um ambiente real aproximado a este. Também nota-se que, como soluções NewSQL também são afetadas pelo contexto, um teste de *benchmarking* pode ressaltar em que contexto, ou com quais tipos de dados cada uma das aplicações escolhidas se mostraria mais útil.

Este trabalho então tem o objetivo de enriquecer a literatura acerca do paradigma NewSQL, revisando conceitos que o cercam, e as características que o compõem. Por fim, fazer uma comparação entre 4 produtos baseados neste paradigma. O experimento deve avaliar se os produtos cumprem as características esperadas, baseado nas definições apresentadas sobre o paradigma NewSQL e ressaltar as diferenças entre cada um deles. Procura-se criar com este trabalho, um documento de suporte á adoção ou estudo sobre o paradigma e produtos com a filosofia NewSQL.

1.2 Objetivos

1.2.1 Objetivo geral

Este trabalho tem como objetivo acrescer a literatura acerca do paradigma NewSQL, discorrendo e analisando quatro soluções de bancos de dados alinhados a este paradigma através de uma análise de *benchmarking*, construindo um comparativo entre os produtos e os resultados observados nos testes.

1.2.2 Objetivos específicos

Para alcançar o objetivo final do projeto, os seguintes objetivos específicos são requeridos:

- Estudar a literatura básica acerca dos paradigmas relacional, NoSQL e NewSQL, com enfoque neste último, de modo a produzir um breve resumo acerca dos paradigmas e dos conceitos acerca deles;
- Realizar a leitura de artigos para obter o estado da arte sobre o paradigma NewSQL, de modo a obter o embasamento teórico sobre o assunto neste trabalho;
- Revisar a documentação técnica das 4 soluções escolhidas para este trabalho, de modo a mostrar as características, diferenciais e peculiaridades sobre cada solução selecionada;
- Revisar a documentação sobre os tipos de *benchmarking* disponíveis no mercado para avaliação de bancos de dados, para que seja possível decidir e justificar a escolha do modelo de avaliação selecionado para este trabalho;
- Definir o plano de testes, com os critérios de avaliação e resultados esperados dos testes;
- Definir a arquitetura do plano de testes;
- Executar os testes estipulados no plano de testes e na arquitetura definidos;
- Fazer a análise dos resultados obtidos na etapa de implementação dos testes;
- Apresentar conclusões sobre o assunto e demais observações.

1.3 Justificativa

Tão importante quanto estar disponível á comunidade, haver material de apoio de fácil acesso para um produto é uma característica muito importante, principalmente quando nos referenciamos a uma tecnologia nova ou/e em ascensão. As análises de benchmarking são importantes para avaliar as vantagens e desvantagens de certos produtos frente a um conjunto de dados. Na documentação atual, porém, comparações que envolvem BDs NewSQL comumente fazem a comparação junto a produtos do paradigma NoSQL e relacional (FATIMA; WASNIK, 2016) (RUDOLL, 2017), o que pode omitir características importantes dos produtos ou salientar características erroneamente frente a uma comparação inadequada.

Sendo assim, a análise proposta neste trabalho tem como objetivo apresentar a execução de testes de *benchmark*, que é um protocolo de testes que tem por objetivo produzir dados de desempenho que podem ser usados para fazer comparações significativas entre os sistemas testados (JELLY; KERRIDGE; BATES, 1994). Neste trabalho especificamente, os testes serão aplicados a produtos que apresentam características do paradigma NewSQL, sob um conjunto de métricas selecionadas que consigam expor a eficiência, na prática, dos benefícios a que o modelo se propõe. Como objetivo secundário, este trabalho visa enriquecer a literatura a cerca do paradigma, e servir como uma de referência pra auxiliar na escolha de soluções.

1.4 Método de pesquisa

Para este trabalho foi necessário buscar o entendimento das soluções apresentadas, os paradigmas de bancos de dados existentes no mercado e os conceitos que os cercam, além do conhecimento sobre alguns tipos de *benchmarks* que existem para aplicação em sistemas de bancos de dados, suas características e aplicações para definição do plano de testes.

Para alcançar estes objetivos e outros citados nos objetivos específicos, foi realizada uma revisão da literatura, com foco no paradigma NewSQL. A relevância do material selecionado foi acordada em conjunto com o orientador e coorientador. Foi realizado também um estudo da documentação técnica fornecida por cada fabricante das soluções utilizadas. Por fim, foi realizada uma análise sobre os tipos e aplicações dos *benchmarks* existentes, de modo a encontrar os melhores candidatos para os objetivos deste trabalho.

Foi desenvolvido um plano de testes para comparar as características relevantes ao paradigma NewSQL e o contexto de cada aplicação, que foi aplicado nos produtos escolhidos. Ao final, a análise da execução deste plano de testes forneceu informações para a construção dos comparativos e embasamento para as conclusões do trabalho.

1.5 Estrutura do trabalho

O presente trabalho se divide em seis capítulos. No primeiro capítulo são apresentados os objetivos gerais e específicos, a justificativa e a metodologia adotada. No capítulo 2 são apresentados conceitos para entendimento geral do contexto do trabalho e as soluções selecionadas para este trabalho, com suas devidas justificativas. No capítulo 3, três obras correlatas ao tema são analisadas.

A parte de implementação do trabalho está no capítulo 4 e contempla a parte técnica, com o ferramental e tecnologias utilizadas, o plano de testes e sua execução.

No Capítulo 5 temos análises e discussões sobre os resultados obtidos e por fim as conclusões no Capítulo 6.

2 Fundamentação teórica

Neste capítulo são apresentados conceitos e contextualizações necessárias para entendimento do trabalho. Na Seção 2.1 e 2.2 há definições acerca de Big Data e Teorema CAP, e na Seção 2.3 há explanações sobre o processo de *benchmarking* de bancos de dados. Na Seção 2.4 há explicações sobre os termos OLTP e OLAP, e na Seção 2.5 a definição e características de bancos de dados NewSQL.

2.1 Big Data

A quantidade e variedade de dados que produzimos diariamente é cada vez maior. Sistemas evoluem e ficam cada vez mais complexos, e geralmente guardam mais dados ao aumentar sua complexidade. *Big Data* é definido na literatura como uma coleção de dados de grande número e complexas operações de transferências ou transações, que necessitam de um bom sistema de gerenciamento ou aplicação para processar esses conjuntos de dados, que os sistemas de gerenciamento de dados atuais têm dificuldade em lidar (SHAMSUDDIN; HASAN, 2016).

Para exemplificar o conceito convém observar exemplos de aplicação, como um sistema usado na bolsa de valores para controlar as cotações, ou um sistema ERP (*Enterprise Resource Planning*) de uma empresa multinacional com milhares de funcionários, em que são feitos controles de gerenciamento de pessoal (contratações, demissões, controle de ponto). Além disso, com o advento da *IoT* equipamentos interligados a esses sistemas (coletores, sensores de peso, impressoras, etc) também geram dados para enriquecer o rastreamento/processamento das operações. A partir disso é possível observar a enorme quantidade de dados que podem ser geradas nestes sistemas.

O volume de dados não deve ser visto como uma característica exclusiva para a definição do termo Big Data. O conceito está também ligado a ideia de retirar valor dos dados que os sistemas já armazenam em suas rotinas comuns. Considerando dois cenários, um cenário de uma pequena e outro uma grande empresa, temos uma grande diferença no volume de dados, mas a análise correta desses dados pode trazer um valor semelhante para ambos os cenários. Na literatura há a concordância em três características que ajudam a definir o conceito de *Big Data*. Esse conjunto foi chamado de 3V's, um termo cunhado por Doug Laney (LANEY, 2001). Estas características são:

1) Volume (*volume*) - São grandes conjuntos de dados e com crescimento exponencial. Por exemplo, um sistema bancário possui os registros de transações de todos seus usuários , que gera um enorme número de registros em um curto espaço de tempo. Além disso, é importante considerar a variável tempo,

pois o hardware e as tecnologias de armazenamento evoluem e baixam de custo, facilitando armazenagem de uma massa cada vez maior de dados.

2) Velocidade (*velocity*)- Essa característica se refere á velocidade com que os dados são gerados. A necessidade do sistema conseguir receber e produzir informação com os dados é cada vez mais necessária, considerando que muitos dos sistemas oferecem informações atualizadas em tempo real, como apps de celular. Para analisar uma fraude de cartão de crédito por exemplo, as transações de cada cartão têm de ser analisadas no momento em que são feitas, e rapidamente comparar com o padrão de transações que o usuário já fez. O único modo de fazer isso é analisando as informações em tempo real, antes mesmo de estruturar e guardar os dados em um BD.

3) Variedade (*variety*)- Hoje, os dados são gerados de diversas formas. São e-mails, áudios, redes sociais, telefones, etc. Esses dados não são anexados em bancos de dados de forma estruturada, por isso recebem o nome de dados não-estruturados. Para se obter vantagem competitiva e, como já mencionado, acompanhar a velocidade de criação dos dados, é necessário analisar os dados não-estruturados junto com os dados estruturados. O conceito de *Big Data* prevê a análise dessa variedade de dados, de modo a agregar mais valor nos resultados.

Na literatura encontram-se autores que acreditam que esses 3 aspectos já sejam suficientes para entender o conceito. Outros autores estipularam mais alguns aspectos a se considerar (PATGIRI; AHMED, 2016). A este trabalho observou-se a necessidade de comentar sobre 2 V's adicionais, que são: (i)veracidade, referente a extrair da massa de dados, apenas os dados confiáveis através de processos e filtros e (ii) valor, no qual se refere a entender primeiramente, quais questões estratégicas o *Big Data* pode responder para o projeto ou empresa ao qual está sendo usado, para obter benefícios que justifiquem o investimento no processo na totalidade.

2.2 Teorema CAP

O teorema CAP (*Consistency, Avaliability and Partition Tolerance*) foi criado por Eric Brewer, e observa que em sistemas distribuídos, não é possível um sistema garantir ao mesmo tempo, características de consistência, disponibilidade e tolerância a falhas (SHIM, 2012). Seria possível garantir apenas duas destas características ao mesmo tempo, renunciando a uma. Para entender o teorema CAP, a seguir são detalhadas as características a ser consideradas.

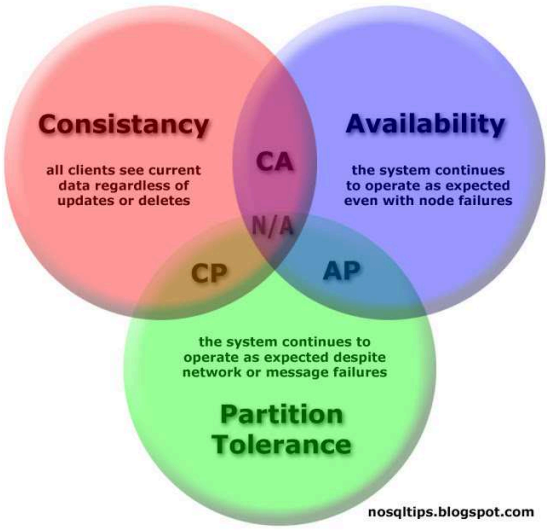
A característica de consistência (C - *Consistency*) infere que, considerando um sistema distribuído, todos os nodos devam ter a mesma visão dos dados. Para garantir essa característica, quando há uma atualização no valor de um dado ou mesmo a inclusão de um novo dado em um dos nodos, nenhuma consulta deve ser executada nos demais nodos até que a alteração/inserção tenha sido replicada para todos.

A segunda característica é a disponibilidade (A - *Availability*). Considerando o contexto de bancos de dados distribuídos, isso quer dizer que o sistema deve operar mesmo sob falhas de hardware ou software, ou ainda sob atualizações de sistema. Geralmente para garantir disponibilidade é criada uma estrutura entre os nodos que considere o balanceamento de cargas de modo a atender o maior número de requisições possíveis.

Por último, a tolerância a falhas (P - *Partition Tolerance*) diz respeito ao modo que o sistema responde e garante continuamente suas funcionalidades. De modo a garantir essa propriedade, geralmente o sistema é criado considerando uma política de replicação total dos nodos. Assim, caso um nodo saia da rede uma réplica toma seu lugar de maneira transparente ao usuário.

Como mencionado anteriormente, um sistema distribuído apenas pode atender simultaneamente duas das três características. Sendo assim temos 3 combinações possíveis do uso do sistema. A Figura 1 mostra as interseções das combinações possíveis e em seguida apresenta-se o detalhamento de cada uma.

Figura 1 – Representação gráfica do Teorema CAP



http://blog.caelum.com.br/nosql-do-teorema-cap-para-paccl/

1) Sistemas CA - Para garantir as características de consistência e disponibilidade (mostrado entre os círculos vermelho e azul na imagem), abre-se mão da tolerância a falha. Sistemas que optam por essa combinação geralmente não

tem particionamento. Exemplos são os bancos de dados relacionais tradicionais (SQL Server, Oracle 11g, PostgreSQL, etc);

2) Sistemas AP - Os bancos que optam por exata combinação (mostrado entre os círculos azul e verde na imagem) tem tolerância a falha, pois, geralmente operam em vários servidores para a aplicação, e tem alta disponibilidade, geralmente ficando com a consistência eventual dos dados. São exemplos a maioria dos bancos NoSQL (Cassandra, Voldemort, etc);

3) Sistemas CP - São sistemas onde há consistência e tolerância a falhas (mostrado entre os círculos vermelho e verde na imagem), mas uma falha em um dos nodos pode deixar o sistema indisponível. Pode acontecer quando nem todos os membros de um cluster se conhecem, ou não há um plano formulado para a queda do master. MongoDB é um exemplo de sistema que pode ser CP, embora o problema possa ser resolvido no gerenciamento do cluster (STEPPAT, 2011).

2.3 OLAP E OLTP

Classicamente as cargas executadas em BDs são classificas em dois tipos: OLAP (*On-line Analytical Processing*) e OLTP (*On-line Transaction Processing)* (STONEBRAKER, 2011b). Cargas OLTP se referem a consulta de rápida execução que envolvam um ambiente transacional. Geralmente são operações que não necessitam da análise de um grande conjunto de dados para serem executadas. Estas operações geralmente, são as mais comuns na utilização de um sistema, pois refletem operações simples realizadas pelo usuário. Em um sistema com grande quantidade de usuários simultâneos estas operações tornam-se altamente críticas.

Os dados acessados por estas operações possuem alto nível de detalhes e são estruturados, normalmente, num modelo relacional normalizado. Seu armazenamento é feito geralmente através de sistemas de bancos de dados tradicionais, com alta frequência de atualização e volatilidade nos dados (são passíveis de modificação e/ou exclusão). Estes dados são utilizados por técnicos, analistas, e usuários da organização.

Em resumo, as cargas OLTP podem ser resumidas em 3 características principais (PAVLO, 2016):

1) Curta duração (ou seja, nenhum usuário fica parado);

2) Acessam um pequeno conjunto de dados usando pesquisas de índices (sem leituras de tabelas completas ou grandes joins distribuídos);

3) Repetitividade (mesmas pesquisas com parâmetros diferentes).

No que lhe concerne, as cargas OLAP são usadas no nível estratégico da organização. São operações que envolvem somas e outras operações de agregação e normalmente executadas sobre um grande conjunto de dados. São otimizadas para leitura e geração de análises e relatórios gerenciais. Seus dados são estruturados em modelagem dimensional, geralmente com alto nível de sumarização.

Seu armazenamento é realizado em estruturas específicas, como um *Data Warehouse*, com otimização no desempenho em grandes volumes de dados. A atualização dos dados tem uma baixa e específica frequência, definida pela organização. Esses dados são utilizados por gestores e analistas para tomada de decisão(ELIAS, 2016).

2.4 Benchmarking

A tarefa de avaliar o desempenho de um sistema de banco de dados não é trivial. Há diversas características que compõem o sistema e otimizações dos fabricantes podem tornar um produto mais atrativo para um domínio de problema do que outros.

Um modo de avaliar e comparar várias características dos produtos existentes é através do processo de benchmarking. Segundo Jim Gray (GRAY, 1993), o *benchmarking* é um teste padronizado que é executado em vários sistemas, e as características de desempenho neste teste são medidas e gravadas. O desempenho geralmente é uma métrica da taxa de transferência (trabalho/segundo), e o preço é normalmente uma métrica de custo total da posse, que é uma estimativa financeira que avalia os custos diretos e indiretos relacionados á compra de um hardware ou software. Juntas, essas duas métricas dão uma relação preço/desempenho.

Entretanto, a visão genérica de *benchmark* apenas proporciona uma relativa relação entre preço e desempenho. Alguns testes mais genéricos de *benchmark* são fornecidos por organizações como a TPC[1], uma organização sem fins lucrativos, que define padrões de *benchmark* para a indústria (MAJDALANY, 2001). Os testes da TPC se popularizaram, e grandes marcas, como *Microsoft* e *Oracle* lançam junto aos seus produtos as pontuações obtidas nestes testes[2,3].

Com o advento dos bancos NoSQL a situação mudou. Os produtos que envolvem o paradigma não relacional são em sua maioria especializada, ou seja, concentram os esforços em certas características, como por exemplo, velocidade de acesso, no caso dos bancos key-value como Redis, ou disponibilidade, como no caso do Cas-

[1] http://www.tpc.org/
[2] https://www.microsoft.com/en-us/sql-server/sql-server-benchmarks-industry
[3] http://www.oracle.com/us/solutions/benchmark/apps-benchmark/benchmark-results-063877.html

sandra. Dessa forma, os testes padrões da indústria não conseguem avaliar essa especificidade, o que deixa uma lacuna por testes de domínio específicos.

Neste trabalho utilizamos de *benchmarks* de domínio específico. Isso quer dizer que cada *benchmark* especifica uma carga sintética caracterizando aplicações típicas do problema de domínio (GRAY, 1993). Os resultados destes testes fornecem uma estimativa do funcionamento relativo no domínio do problema e oferecem um resultado mais próximo da realidade, do que testes que utilizam dados e volumes genéricos de teste. Mais detalhes sobre os *benchmarks* utilizados neste trabalho são vistos nas seções 4.1.1, 4.1.4 e 4.1.2.

2.5 Bancos de dados NewSQL

Hoje sistemas tentam agregar o máximo de informação na organização, de modo a integrar, automatizar ou manter registro de todas as possíveis atividades de uma companhia. Os sistemas de bancos de dados relacionais foram pensados em um diferente momento, e hoje encontram desafios em atender aos requerimentos de desempenho e escalabilidade necessários a suprir essa grande quantidade de dados, muitas vezes denominada como *Big Data* (KARAMBIR; MONIKA, 2017). Acredita-se que questões como estas inspiraram o movimento NewSQL. O termo NewSQL foi cunhado por Michael Stonebreaker (STONEBRAKER, 2011b). Os bancos de dados NewSQL são classificados como uma classe de modernos SGBDs relacionais que buscam oferecer o mesmo desempenho escalável dos bancos NoSQL, para cargas de trabalho OLTP *read-write*, mantendo as garantias das propriedades ACID em suas transações (PAVLO, 2016).

Os bancos NewSQL são particularmente interessantes para se utilizar como um serviço (*Database-as-a-service - DBaaS*), considerando que os produtos baseados nessa tecnologia já recebem as consultas SQL via requisição a um serviço. Caso estes bancos sejam implementados em *datacenters*, como os serviços oferecidos pelas grandes marcas *Google* e *Amazon*, podem ser escalados com certa facilidade. Desta forma é possível iniciar novos nodos conforme a necessidade, sendo assim um banco de dados elástico e podendo resultar em redução dos custos com infraestrutura.

Em outra definição da literatura, o termo NewSQL é definido como uma classe de modernos sistemas de gerenciamento de bancos de dados (SGBDs) que provém o mesmo desempenho escalável dos bancos de dados NoSQL para cargas OLTP, enquanto garante as propriedades ACID dos sistemas de bancos de dados tradicionais (PAVLO, 2016). Tais características têm o propósito de resolver problemas de escalabilidade, desempenho e disponibilidade observados nos sistemas de banco de dados relacionais tradicionais, ao mesmo tempo, que provém a capacidade de consulta

via SQL (KARAMBIR; MONIKA, 2017). Os bancos deste paradigma de dados se mostram assim bons candidatos para substituir implementações de NoSQL, ou atualizar interfaces antigas que utilizam o sistema relacional via SQL.

2.5.1 Características de um banco de dados NewSQL

Além de oferecer a SQL como linguagem primária de interação e as garantias ACID como vantagens (KARAMBIR; MONIKA, 2017), Pavlo e Aslett (PAVLO, 2016) definem algumas características a que o paradigma NewSQL deve atender e em que se diferencia dos modelos tradicionais. São elas:

2.5.1.1 Uso da memória principal

Diferente dos sistemas tradicionais que utilizam a persistência totalmente em disco, utilizando somente a memória para otimizar certas partes do processo, os produtos NewSQL geralmente utilizam a memória principal (RAM) como memória principal para o banco, com a finalidade de evitar ao máximo o uso de discos rígidos. O benefício dessa abordagem é habilitar certas otimizações, visto que o sistema não necessita de mecanismos para lidar com a troca de informações entre disco e memória constantemente.

Apesar de constar como característica de um paradigma relativamente novo, a ideia de utilizar o banco todo em memória não é nova, já sendo mencionada em meados dos anos 80 (GARCIA-MOLINA; LIPTON; VALDES, 1984; DEWITT et al., 1984). A diferença para os sistemas atuais está na habilidade de desalocar um subconjunto da base de dados para memória persistente para reduzir seu consumo de memória. Isso permite que o banco de dados suporte bases de dados maiores do que o total de memória disponível no sistema, sem ter de voltar a arquitetura orientada a discos rígidos. A abordagem mais usada para isso é criar um mecanismo de monitoramento que identifica quais tuplas não estão sendo acessadas e podem ser desalocadas da memória. Em seu lugar é deixado um ponteiro no local para referência. Quando esse dado é referenciado, de forma assíncrona uma *thread* separada carrega os dados de volta para a memória.

2.5.1.2 Particionamento/*Sharding*

A forma que quase todos os SBGD's NewSQL escalam é dividindo a base de dados em subconjuntos, chamados de partições ou *shards.* A ideia também não é nova, tendo como primeiros produtos as versões distribuídas dos sistemas System R e INGRES em 1980. Nesta arquitetura, as tabelas são divididas horizontalmente entre múltiplos fragmentos, limitados pelos valores de uma ou mais colunas da tabela.

O SGBD designa cada tupla a um fragmento, baseado nos valores desses atributos usando particionamento de intervalo ou hash.

Um aspecto novo no particionamento, porém, é de que há suporte a migração de tuplas sem que o sistema fique indisponível (*offline*). Isso quer dizer que o SGBD pode mover dados entre estrutura física, reequilibrando e aliviando pontos de acesso, ou aumentar / diminuir a capacidade do SGBD sem interrupção do serviço. Essa funcionalidade é presente em alguns produtos NoSQL, mas nesta nova classe de bancos de dados agrega as garantias ACID no processo.

2.5.1.3 Controle de concorrência

O controle de concorrência é citado como o detalhe mais importante de um sistema gerenciador de banco de dados, pois afeta praticamente todos os aspectos de um sistema (PAVLO, 2016). O controle de concorrência é o que permite que vários usuários acessem um sistema ao mesmo tempo, e tenham a ilusão de que estão executando suas transações num sistema dedicado. Isso fornece essencialmente as garantias de atomicidade e isolamento no sistema e, como tal, influencia todo o comportamento o seu comportamento. No final da década de 70, o SDD-1 foi o primeiro banco lançado especificamente desenhado para transações sob um cluster através de uma arquitetura *shared-nothing* controladas por um coordenador central (BERNSTEIN et al., 1978).

Uma parte expressiva dos sistemas NewSQL utilizam de uma variante do controle de concorrência baseado em *timestamp ordering* (TO), na qual o SGBD assume que as transações não executarão operações intercaladas que violarão a ordem serializável. O protocolo mais amplamente utilizado nos sistemas NewSQL é o MVCC (*descentralized multi-version concurrency control*), no qual o SGBD cria uma versão de uma tupla no banco de dados quando é atualizada por uma transação. A manutenção de várias versões permite potencialmente que as transações ainda sejam concluídas, mesmo se outra transação atualizar as mesmas tuplas. Ele também permite que transações de longa duração, somente leitura, não bloqueiem os escritores.

2.5.1.4 Índices secundários

Um índice secundário é um subconjunto de atributos de uma tabela, mesmo não sendo uma chave primária. Isso permite que o SGBD ofereça suporte a consultas rápidas buscando em outros atributos além das chaves primárias, que criam índices naturalmente nos sistemas existentes. Isso permite que o SGBD ofereça suporte a consultas rápidas além da chave primária ou ao particionamento de pesquisas importantes. Isso é mais facilmente implementado em sistemas não particionados,

devido aos dados se encontrarem no mesmo local. Já em bancos distribuídos, estes índices nem sempre podem ser particionados da mesma maneira que o resto do banco. Por exemplo, se uma tabela de clientes for particionada baseado em sua cháve primária (id), e há consultas que querem consultar o id do cliente baseado no e-mail, o SBGD distribuído terá de transmitir essas consultas para cada nó, o que é ineficiente. Então, para suportar índices secundários, duas decisões de *design* são importantes: (i) onde o sistema irá armazená-los e (ii) como irá mantê-los no contexto das transações.

Os produtos NewSQL são decentralizados e usam índices secundários particionados. Isso quer dizer que cada nodo guarda uma porção do índice. O *trade-off* entre índices particionados e replicados é que, com as consultas anteriores, pode ser necessário abranger vários nós para encontrar o que eles estão procurando, mas se uma transação atualiza um índice, ele terá apenas que modificar um nó.A maneira mais comum pela qual os desenvolvedores criam índices secundários ao usar um DBMS NewSQL que não os suporta é implantar um índice usando um *cache* distribuído na memória.

2.5.1.5 Replicação

Um modo de uma organização garantir alta disponibilidade e durabilidade dos dados nas aplicações OLTP é replicando sua base. Os bancos modernos, incluindo a classe dos NewSQL, suportam algum tipo de mecanismo de replicação. No caso dos *DBaaS*, há uma vantagem distinta nessa área, pois ocultam todos os detalhes da configuração de replicação de seus clientes. Isso facilita a implantação de um SGBD replicado sem que o administrador tenha que se preocupar com a transmissão de *logs* e garantir que os nós estejam sincronizados.

Os sistemas NewSQL devem suportar replicação fortemente consistente (PAVLO, 2016). O modo de garantir essa consistência, pela replicação de máquinas de estado, já são estudados desde meados dos anos 70 (GRAY, 1978).

Um aspecto dos sistemas NewSQL diferente dos métodos tradicionais é a consideração de replicação sobre a rede de longa distância (WAN). Esse é um subproduto de ambientes operacionais modernos, onde agora é trivial implantar sistemas em vários *datacenters* separados por grandes diferenças geográficas. Qualquer DBMS NewSQL pode ser configurado para fornecer atualizações síncronas de dados pela *WAN*, mas isso causaria lentidão significativa nas operações normais. Assim, eles fornecem métodos de replicação assíncrona.

2.5.1.6 Tolerância a falhas

Outra característica importante de um DBMS NewSQL para fornecer são os mecanismos de recuperação em casos críticos. A tolerância a falhas é a capacidade do sistema lidar com uma falha na estrutura. Ao contrário dos SGBDs tradicionais, onde a principal preocupação da tolerância a falhas é garantir que nenhuma atualização seja perdida (MOHAN et al., 1992), os SGBDs da classe NewSQL devem minimizar o tempo de inatividade, pois se espera que os aplicativos modernos estejam *online* o tempo todo e que as interrupções no site não sejam dispendiosas.

Considerando um sistema distribuído com réplicas, quando o mestre fica *offline* por algum motivo, outro mestre toma seu lugar. Nesse tempo as transações continuam a ser recebidas e processadas. Caso o antigo mestre retorne, ele primeiramente precisa obter as atualizações do novo mestre (e potencialmente outras respostas) que perdeu enquanto estava inativo, e só então estará apto a retomar o lugar como mestre, o que não necessariamente precisa ocorrer.

2.5.2 Vantagens e desvantagens

Existem pontos a se considerar para a adoção da nova classe de bancos de dados NewSQL (PAVLO, 2016). Entre as vantagens, estão:

1) Os bancos de dados NewSQL são construídos para execução distribuída - Nesta classe de banco de dados, todas as partes do sistema podem ser otimizadas para ambientes multi-nodos. Isso inclui partes como o otimizador de consultas e o protocolo de comunicação entre nodos.

2) Gerenciamento próprio do *storage* primário - Com exceção do produto Google Spanner, os bancos de dados NewSQL são responsáveis por distribuir o banco de dados por entre seus recursos com um mecanismo personalizado, em vez de depender de um sistema de arquivos distribuído pronto para uso (por exemplo, HDFS) ou de uma estrutura de armazenamento (por exemplo, Apache Ignite).

Em contrapartida, como principal desvantagem desta classe de bancos de dados está a preocupação de muitas empresas na adoção destas tecnologias, devido ao fato de ser uma tecnologia nova e pouco analisada para embasar sua utilização em grandes bases de produção. Isso significa que o número de pessoas que estiveram em contato com o sistema ainda é muito pequeno, comparando com os grandes nomes de SGBDs tradicionais. Além disso, uma organização provavelmente abrirá mão do acesso às ferramentas de administração e controle, criados para os sistemas mais populares

(PAVLO, 2016, 47). Algumas das soluções NewSQL resolvem esse problema mantendo compatibilidade com sistemas legados, como MySQL (Clustrix e MemSQL).

2.5.3 Produtos

Nesta seção são explanadas as características principais referentes aos produtos selecionados para a análise neste trabalho. Os critérios de escolha se basearam na adequação ao paradigma NewSQL dos produtos, e uso da iniciativa de código aberto e popularidade na comunidade, e pelos produtos disponibilizarem uma versão sem custo, em que fosse possível avaliar suas características. A escolha final foi pelos produtos VoltDB[4], NuoDB[5], CockroachDB[6] e MemSQL[7].

2.5.3.1 VoltDB

O VoltDB é um sistema de banco de dados desenvolvido por uma empresa que carrega o mesmo nome. Foi desenvolvido em 2014 pelos cientistas da computação: Dr. Michael Stonebraker, também criador do termo NewSQL, Sam Madden, e Daniel Abadi. Ela é disponibilizada em versões *enterprise* e *community*, sendo esta última sob licença *GNU Affero General Public License* (VOLTDB, INC, 2018).

A arquitetura do produto VoltDB pode alcançar até 45 vezes mais taxa de transferência do que as soluções de bancos de dados tradicionais. A arquitetura permite escalar facilmente, adicionando processadores ao clusters dinamicamente conforme o volume e necessidades de transações crescem (VOLTDB, INC, 2018).

O funcionamento geral do banco VoltDB utiliza as seguintes características:

- Uso de armazenamento em memória principal para maximizar a vazão de dados, previnindo custosos acessos em disco aos dados;
- Ganho de desempenho por serialização de acesso a todos os dados, previnindo o consumo de tempo das funções de *locking*, *latching*, logs de transação, etc, dos tradicionais SGBDs;
- Clusterização com replicação sob múltiplos servidores, que garantem escalabilidade, confiabilidade e alta disponibilidade dos dados.
- Compatível com as características ACID e uso da ANSI *standard* SQL, que garante uma rápida curva de aprendizado dos usuários avançados de bancos de dados e permite fazer as transações direto da aplicação.

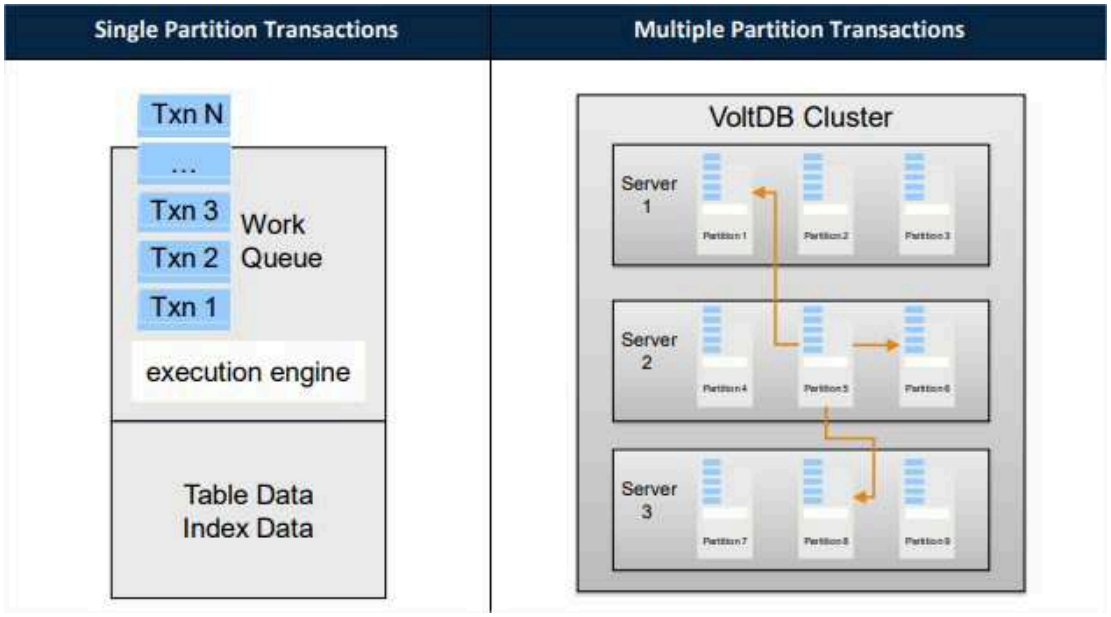
[4] https://www.voltdb.com/
[5] https://www.nuodb.com/
[6] https://www.cockroachlabs.com/
[7] https://www.memsql.com/

A documentação do produto informa que o produto VoltDB não é otimizado para todos os problemas envolvendo bancos de dados. O segmento de mercado a que atende são aplicações que necessitam processar grandes quantidades de dados rapidamente, como aplicações financeiras, redes sociais, etc. Nestes sistemas, os requisitos são escalabilidade, confiança alta disponibilidade e uma ótima taxa de transferência. Aos casos como coletar e agrupar conjuntos de dados históricos em bases extremamente grandes por exemplo, o fornecedor menciona que não seria o melhor uso do produto, e indica procurar outros produtos mais direcionados (VOLTDB, INC, 2018).

2.5.3.1.1 Funcionamento

A Figura 2 mostra uma visão geral da arquitetura do VoltDB:

Figura 2 – Componentes da arquitetura VoltDB



http://www.odbms.org/wp-content/uploads/2013/11/VoltDBTechnicalOverview.pdf

Na Figura2, um exemplo de cluster VoltDB é exposto. Na parte da direita, temos a figura do cluster. O cluster possui servidores, delimitados por retângulos na figura, e dentro deles processadores, que armazenam partições do Volt. Um olhar mais atencioso dessas partições é feito na parte a esquerda da figura, onde podemos observar as partes de uma partição, composta pela fila de processamento (*Work Queue*), toda a *engine* de execução (*Execution Engine*) e as tabelas e dados indexados (*Table and Index Data*). Por fim, na parte direita pode se observar as setas interligando os servidores, que representam a comunicabilidade entre o cluster na situação em que há uma consulta que necessita dados de múltiplas partições. Neste cenário, um dos nodos age como coordenador e distribui o trabalho necessário entre os outros

nodos, coletando os resultados e completando a tarefa (VOLTDB, 2013). A seguir, os componentes que compõem o VoltDB são explanados.

- **Particionamento** - No VoltDB cada *procedure* armazenada é definida como uma transação (*Work Queue* na Figura 2). Ela pode ser bem sucedida, ou ocorrer um rollback, para garantia da consistência. O VoltDB analisa e pré-compila o acesso lógico que a procedure vai precisar, e distribui os dados e processamento associados com as partições desejadas no clusters. Desse jeito, cada partição fica com uma "fatia única" do processamento. Cada nodo também suporta diversas transações.

- **Processamento de Transações** - Em tempo de execução na *Execution Engine*, as chamadas a procedures são passadas para as partições apropriadas. Quando as procedures são "*single-partitioned*", significa que ela opera os dados dentro de uma única partição, o servidor executa a procedure na partição, liberando o resto do cluster para realizar outras requsições em paralelo. Isso permite operar sem locks e escalar facilmente. Para transações que necessitam dados de múltiplas partições, um nodo age como o coordenador, e gerencia o trabalho necessário aos outros nodos, coletando o resultado e completando a tarefa. Essa coordenação faz as transações distribuídas serem um pouco mais lentas que as não particionadas. Porém, a integridade transacional é mantida e a arquitetura consegue suportar múltiplas operações paralelas.

- **Particionamento e replicação de tabelas** - Tabelas são particionadas no VoltDB baseado na coluna que o desenvolvedor ou *designer* especifica, que podem selecionar colunas mais acessadas pelas procedures. Essas colunas são otimizadas pelo VoltDB em tempo de execução. O VoltDB também permite replicar tabelas para todas as partições no cluster, o que pode ser necessário para tabelas pequenas mas muito acessadas. O otimizador de consulta do VoltDB pode verificar dinâmicamente a consulta e criar cópias das tabelas mais usadas para otimizar o desempenho na leitura caso ela esteja sendo muito usada para operações de leitura, por exemplo.

- **Escalabilidade** - O VoltDB escala para mais nodos sem tempo de espera, e, segundo a documentação, sem sacrificar o tráfego de rede. Ao unir mais um nodo ao *cluster*, ou remover, o VoltDB rapidamente distribui a demanda entre os nodos, começando pelos conteúdos mais acessados, num esquema que a empresa chama de "*no wait architecture*".

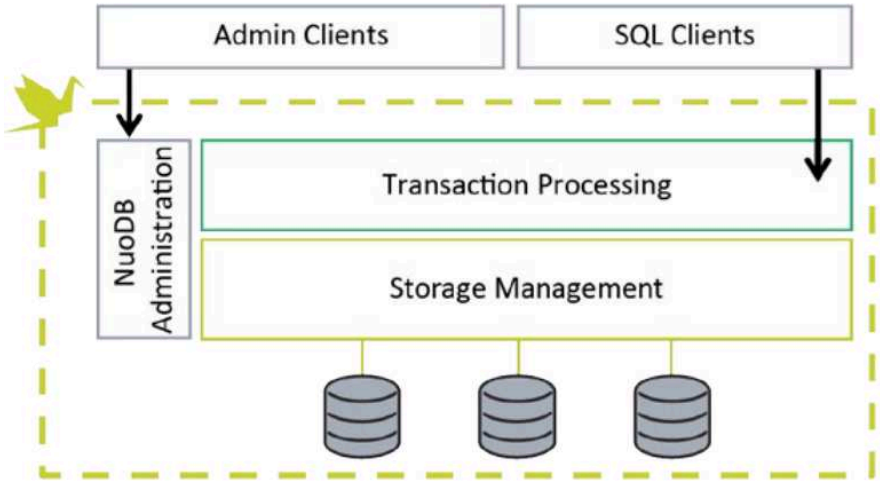
2.5.3.2 NuoDB

O NuoDB foi criado em 2008 por Jim Starkey e Barry Morris. Sua primeira versão foi lançada em 2010. A empresa leva o mesmo nome do produto. O produto é disponibilizado em 3 versões: *Community*, que é gratuita, sob licença proprietária; *Professional* e *Enterprise*. A versão Community utilizada no trabalho inclui restrições de escalabilidade: Falando nos termos da arquitetura do NuoDB, a versão grátis permite 1 instância de administração, 1 SM(*Storage Management*) e 3 TE's (*Transaction Processing*) para escalabilidade (na versão paga a escalabilidade pode ser r por ilimitados SM's e TE's). Há também outras vantagens menores relacionadas á experiência do usuário, como acesso à interface gráfica de gerenciamento e suporte diferenciado do produto (NUODB INC., 2018b). O produto recomenda para operação em ambiente de produção o sistema *Red Hat Enterprise Linux (RHEL)/CentOS 7.0 (x64)* ou superior, ou RHEL/CentOS 6.5 (x64) ou superior.

A aplicação mais citada do produto pelo fornecedor é em ambientes de nuvem, em forma de *DBaaS*. O fornecedor cita que o produto se comporta logicamente para o usuário final como um ambiente único, e fornecendo características ACID nas transações e consultas via SQL no padrão *ANSI-Standard*(NUODB INC., 2018a). A estrutura interna do NuoDB se propõe a escalar elasticamente, e o produto dispõe de ferramentas de orquestração para gerenciar e administrar o banco de dados e o ambiente da aplicação.

Como disposto na própria documentação, o produto suporta processamento do tipo HTAP (*Hybrid Transactional and Analytical Processing*), que é um termo novo, aplicado a produtos que conseguem atender a cargas OLTP e OLAP ao mesmo tempo (COELHO et al., 2017). O produto aparece para a aplicação como uma base lógica e única via SQL. Contudo, ao nível de tecnologia, o NuoDB utiliza uma tecnologia que chama de *Durable Distributed Scale-out*, uma arquitetura moderna, onde o NuoDB utiliza serviços separados para processamento de transações e gerência do armazenamento. Essa segunda camada possibilita um processamento distribuído que pode ser implantado sob múltiplos *datacenters* e é otimizado para velocidades em memória, disponibilidade contínua e crescimento elástico. A seguir, uma representação da arquitetura do NuoDB:

Figura 3 – Visão geral da arquitetura NuoDB



http://go.nuodb.com/rs/nuodb/images/Technical-Whitepaper.pdf

Na Figura 3 podem ser observadas as duas camadas introduzidas anteriormente. A seguir, os componentes desta arquitetura são detalhados.

- ***Transaction processing layer*** - A camada que recebe as requisições SQL, como mostra a Figura 3, é uma camada que contém nodos em memória chamados *Transaction Engines* (TE's). Essas TE's permitem manter desempenho alto em memória. Quando uma aplicação faz requisições ao NuoDB, os TE's criam *cache* em memória para a carga de trabalho da aplicação. As requisições para arquivos que não estão em *cache* são alimentadas com *caches* de memória de outros TE's ou pela camada de gerenciamento de armazenamento.
- ***The storage management layer*** - A segunda camada mostrada na imagem conectando com os dados físicos, consiste em nodos de processo chamados *Storage Managers* (SM's), que possuem tanto componentes em memória quanto guardados em disco. Os SM's também oferecem garantias de durabilidade dos dados. Múltiplos SM's podem ser usados para aumentar a redundância de dados.

As duas camadas acima descritas tornam o *NuoDB* escalável, pois necessita simplesmente adicionar ou remover TE's e SM's para dimensionar o banco conforme a necessidade.

Sobre a arquitetura distribuída, a documentação descreve que a arquitetura de duas camadas do *NuoDB* permite uma situação que é descrita como "*Active-active*". Ele pode ser implementado em um *datacenter*, sob muitos data centers ou sobre uma nuvem híbrida, mantendo de qualquer forma as garantias ACID e velocidade de acesso

em memória. Se um *datacenter* sofrer um desastre, o resto da base pode perfeitamente assumir a carga de trabalho sem nenhum *downtime*. A complexidade do balanço de carga, conflitos e outros problemas que podem vir dessa transição, são gerenciados automaticamente pela aplicação.

2.5.3.3 CockroachDB

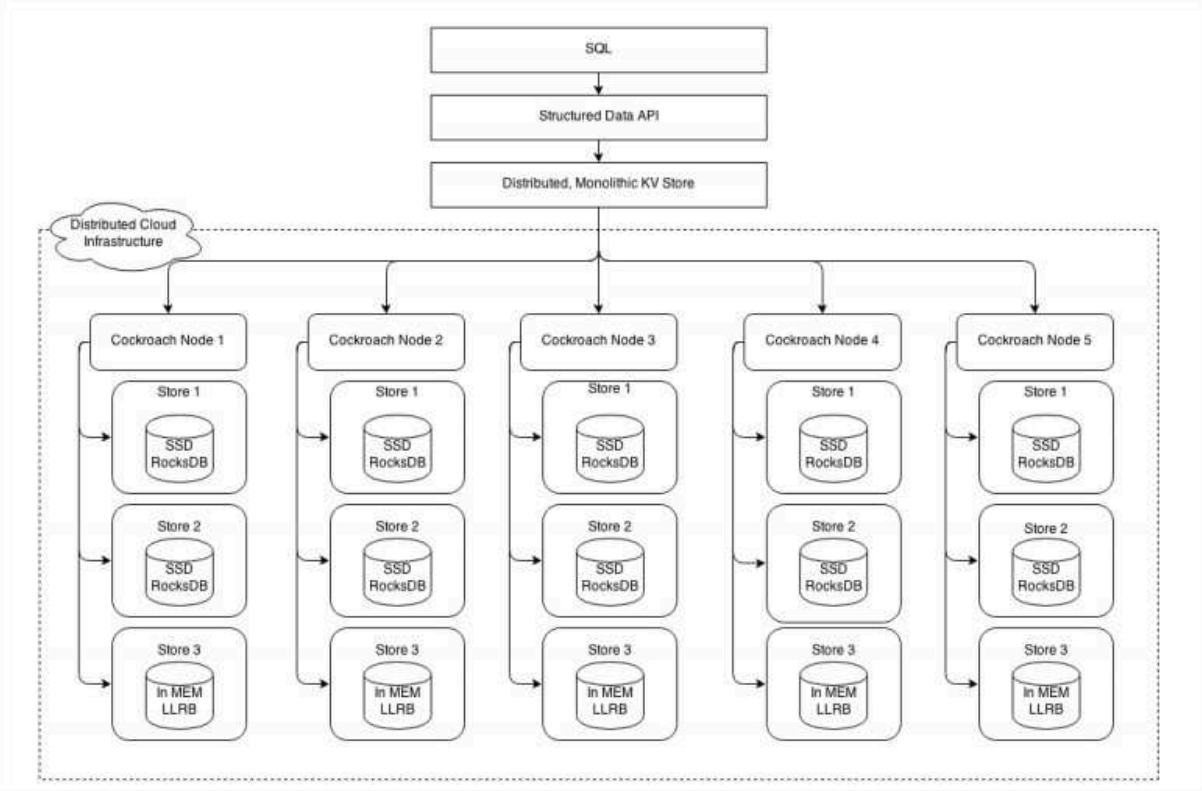
O *Cockroach DB* é um sistema de banco de dados criado pela empresa Cockroach Labs em 2015. É formada por três ex-funcionários do Google: Spencer Kimball, Peter Mattis, and Ben Darnell. Segundo a documentação (LABS, 2018), o projeto Cockroach DB foi criado para ser um banco de dados *open-source,* distribuído no nível de que uma instância pode ser levantada em um computador pessoal comum e ajudar no processamento das requisições feitas.

O produto é distribuído nas versões *Core* e *Enterprise*, sendo a primeira grátis. A diferença entre as versões ficam em que a versão *Enterprise* permite geoparticionamento, permissões especiais para usuários, serviços melhores de monitoramento do cluster e suporte técnico. O programa, como já mencionado, tem código aberto para as duas versões. A versão *Core* é disponibilizada “grátis para sempre”.

A ferramenta se define como um banco de dados *open-source,* consistente e escalável globalmente. Apesar de novo entre os bancos de dados do paradigma NewSQL, ele não utiliza armazenamento final em memória principal. Ao invés disso, é feito o aproveitamento de uma estrutura de clocks atômicos para escrita de blocos, que facilita a obtenção das características ACID nas transações.

A arquitetura converte comandos SQL em dados *Key-Value* (KV), que são extremamente rápidos de manipular. Essa arquitetura é composta de 6 camadas que interagem diretamente com as camadas acima e abaixo de cada uma. A Figura 4 representa a arquitetura do CochroackDB:

Figura 4 – Visão das camadas do CockroachDB



http://cs.ulb.ac.be/public/_media/teaching/cockroachdb_2017.pdf

Na imagem é possível ter uma visão hierárquica das camadas que compõem o produto. A seguir são detalhadas as partes que compõem essa arquitetura.

1) *SQL Layer* - Mostrada na figura 4 , a camada SQL recebe as consultas ao banco de dados via uma interface API, para qualquer nodo do cluster, uma vez que todos se comportam de maneira simétrica. A camada SQL repassa essa mensagem para a camada de estruturação, que será responsável por criar o plano de execução.

2) T*ransaction Layer* - A camada de transação, mostrada na figura 3 como *Structured Data API*, é responsável por criar o plano de execução para as requisições SQL recebidas. A consulta e transformada num plano de execução. O plano é passado para a camada subsequente, um gerenciador de operações. Para prover consistência, as características ACID são garantidas na camada de transação. Todas as declarações são tratadas como transações, incluindo declarações simples, pois o comportamento é como se toda declaração fosse seguida de um COMMIT. Para poder enviar uma transação para o *cluster* todo, as transações passam por um processo de *commit* em duas fases. Na primeira fase, é criado um registro de transação quando a escrita ocorre. Esse registro é

guardado em memória. São criadas intenções de escrita (*Write Intents*) para os nodos. A segunda fase consiste no *commit*, onde, após as intenções de escritas serem aceitas, ocorre a efetivação do registro de transação.

3) *Distribution Layer* - Englobando na figura 4 a camada *Distributed, Monolothic KV Store* e a camada *Distributed Cloud Infrastructure* e englobando a *Distribution e Replication Layer,* a camada de distribuição é responsável por guardar um mapa classificado monolítico com os pares de chaves *Key-Value*, um espaço que descreve todos os dados do *cluster* e sua localização. O mapa é dividido em "intervalos", de modo que a consulta das chaves não fique centralizada em um nodo. No mapa, encontra-se a localização dos dados por nodo, e as consultas então são distribuídas entre os nodos,que executam as consultas e recuperam os resultados para as camadas superiores.

4) *Replication Layer* - Essa camada é responsável por copiar os dados entre os nodos, e aplicar o algoritmo de "*consensus*" para garantir a consistência. Para garantir essa consistência, o algoritmo de "*consensus*" funciona de forma que necessite um quórum de nodos ativos para aceitar uma transação. O menor número possível para essa funcionalidade são três nodos ativos. Ao começar a haver falhas na replicação, o CockroachDB verifica se o número de nodos falhando é menos que o fator de replicação/2 - por exemplo, se há 4 nodos, pode haver duas falhas. Ocorrendo mais que isso, o trabalho é redistribuído para os nodos em fucionamento para garantir o funcionamento.

5) *Storage Layer* - Na camada de armazenamento (*Cockroach Nodes* na Figura 4), cada nodo do CockroachDB possui ao menos um *store*, que é o espaço onde o processo do DB lê e escreve dados no disco. São guardados os dados através da API RocksDB[8], que guarda os valores *key-value* no disco. Cada instância do Cockroach possui três instâncias do RocksDB: Uma para o *log*, uma para guardar os dados temporários SQL e a última para todos os dados no nodo. Em adição a essas camadas, em todos os nodos há um bloco de *cache* compartilhado entre os *stores* em um nodo, onde é guardada uma coleção de cópias de partes dos dados do nodo, para segurança.

Cada uma destas camadas que compõem a arquitetura do CockroachDB se comunica com a camada imediatamente inferior. A ligação nas camadas da arquitetura permite que o comando SQL chegue no armazenamento de chave-valor com sucesso.

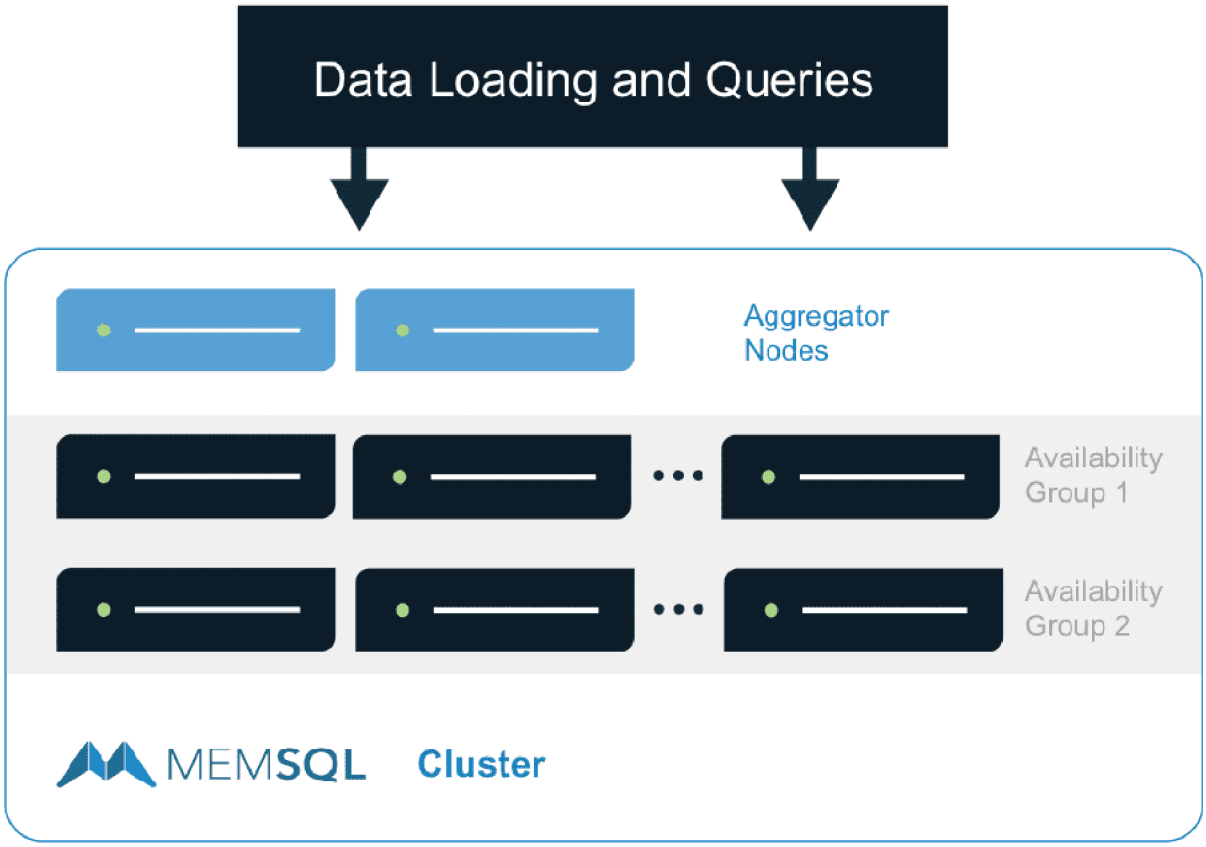
[8] https://rocksdb.org/

2.5.3.4 MemSQL

MemSQL é um sistema gerenciador de banco de dados distribuído e *in-memory*, distribuído pela empresa MemSQL Inc. O primeiro *release* do produto foi em 23 de abril de 2013. É escrito na linguagem de programação C e distribuído nas versões Developer, que é grátis, mas não é recomendado o uso em produção e tem recursos limitados, e a versão *Enterprise*, paga, com todas as funcionalidades. Atua sob licença privada.

O MemSQL é apresentado pelo fornecedor como um banco de dados relacional distribuído que suporta transações e análises em tempo real, em escala. É acessível pela sintaxe SQL, incluindo *joins*, filtros e capacidades analíticas (agregações, funções de janelamento, *group by*, etc). Ele escala horizontalmente e mantém compatibilidade com tecnologias do ecossistema de processamento de dados (plataformas de orquestração, IDE's, e ferramentas de BI). Na Figura 5 é apresentada uma visão da arquitetura do MemSQL.

Figura 5 – Visão simplificada da arquitetura do MemSQL



https://www.memsql.com/content/architecture/

A estrutura do MemSQL é basicamente composta de duas camadas: Os nodos

agregadores (*Aggregator nodes*) e os nodos folha (*Leaf nodes*). Os nodos agregadores, mostrados na parte superior da Figura 5, funcionam como roteadores de consulta, e atuam como um *gateway* no sistema distribuído. Eles armazenam apenas metadados e dados de referência, e distribuem as consultas nos nós folha e agregam os resultados enviados de volta ao cliente. Uma forma de melhorar operações como o carregamento de dados e permitir processar mais solicitações simultaneamente é aumentar o número de agregadores (MEMSQL, 2018).

As cargas de trabalho das aplicações podem ser divididas entre grupos de agregadores (*Availability Groups*), mostradas mais ao centro na figura 5. Um grupo de agregadores pode servir a aplicação A, enquanto outro grupo a aplicação B, e assim por diante.

Os nodos folha (*Leaf nodes*) não são mostrados na Figura 3. Possuem a função de armazenar e computar as tarefas. Os dados são automaticamente distribuídos através dos nodos folha para as partições que fazem a execução paralelizada das consultas. Aumentar o número de nodos folha no sistema aumenta a velocidade de execução das tarefas, especialmente as que necessitam grandes consultas e agregações, e aumenta a capacidade de paralelismo do banco.

A comunicação entre os nodos é realizada via comandos SQL sob um protocolo MySQL[9]. As duas camadas da arquitetura possuem planos de recuperação automáticos em caso de falha para prover tolerância a falhas. A proporção de nodos agregadores e nodos folha na aplicação determina a capacidade e o desempenho do cluster, e pode variar conforme o domínio da aplicação ao qual se designam (MEMSQL, 2018).

2.5.3.5 Outras opções não avaliadas

Na elaboração deste trabalho, foram selecionados mais produtos que atendiam aos nossos critérios, e cabe fazer um comentário sobre dois dos produtos não mencionados. Os produtos Google Spanner[10], hoje parte da arquitetura de serviços *cloud* na plataforma Google One, e o produto Clustrix[11], que foram considerados populares na revisão bibliográfica deste trabalho (PAVLO, 2016; KARAMBIR; MONIKA, 2017; GUREVICH, 2015), não foram incluídos. No caso do Google Spanner, não há versão ou período de testes existentes para o produto, e o preço de se utilizar para experimentação não se justificou. No caso do Clustrix, a versão de avaliação existe no site oficial do produto, mas não é acessível diretamente, sendo necessário requisitar um contato com um consultor para iniciar os testes, tentativa que foi utilizada, mas sem resposta

[9] https://dev.mysql.com/doc/internals/en/client-server-protocol.html
[10] https://cloud.google.com/spanner/
[11] https://www.clustrix.com/

por parte da empresa.

2.5.3.6 Comparativo dos produtos

De modo a sintetizar as descrições dos produtos mostrados nesta seção foi criada a Tabela 2.5.3.6, abrangendo características mencionadas para cada produto nas subseções anteriores e agregando informações dos artigos utilizados na análise da bibliografia (PAVLO, 2016, 47) e sites especializados[12]:

Na Tabela 2.5.3.6, podemos comparar os aspectos relativos aos sistemas, com as características dando nome às colunas dos produtos dispostos nas linhas. Foram dispostas informações adicionais ao texto, sobre o controle de concorrência adotado, no caso do trabalho ou *multi-version concurrency control* MVCC ou TO (*Timestamp Ordering*), métodos de acesso ao banco de dados e outras características relevantes.

[12] https://db-engines.com/en/system/MemSQL%3BNuoDB%3BVoltDB

	Ano de lançam.	Memória RAM como memória principal	Controle de concorrência	Compat. com drivers existentes	APIs e outros métodos de acesso	Métodos de replicação	Permite chaves estrangeiras	Detalhes da Arquitetura
CockroachDB	2014	Não	MVCC	PostgreSQL	JDBC, ODBC	Master-Master	Sim	Baseada em armazenamento chave/valor. Usa clocks híbridos par a replicação por rede.
MemSQL	2012	Sim	MVCC	MySQL	JDBC, ODBC	Master-slave	Não	Distribuída, com suporte ao protocolo MySQL
NuoDB	2013	Sim	MVCC	Não	JDBC ODBC ADO.NET	Gerenc. pelo próprio NuoDB	Sim	Arquitetura dividida, utilizando de vários gerenciadores de transação e um gerenciador de armazenamento
VoltDB	2008	Sim	TO	Não	Java API, RESTful API, JDBC	Master-master, Master-slave	Não	Mecanismos de execução single-thread por partição. Suporta operadores de streaming

O autor(2018)

3 Trabalhos relacionados

Este capítulo apresenta um resumo de três trabalhos que foram usados como fundamentação para este trabalho. Os mesmos buscam explanar os conceitos ligados ao paradigma NewSQL e aplicações dos produtos. A análise considera o artigo teórico “What’s Really New with NewSQL”, que apresenta uma visão teórica sobre os BDs NewSQL existentes , e as comparações experimentais nos trabalhos de (GUREVICH, 2015) e (KAUR; SACHDEVA, 2017).

3.1 What’s Really New with NewSQL?

Em um dos primeiros trabalhos que aborda o paradigma NewSQL (PAVLO, 2016), uma análise do histórico dos sistemas de banco de dados, e a crescente necessidade de armazenamento da informação é realizada. Há informações chaves como a introdução da ideia de que a aplicação e os dados deveriam rodar em separado, e as dificuldades na adaptação das empresas em servir dados na velocidade que as aplicações *Web* que foram surgindo necessitaram.

Os autores propõem que o domínio das aplicações que se beneficiariam do paradigma NewSQL são definidas, possuindo transações que possuem as seguintes características:

- São de curta duração (ou seja, nenhum usuário fica parado);
- Acessam um pequeno conjunto de dados usando pesquisas de índices (sem leituras de tabelas completas ou grandes joins distribuídos);
- São repetitivas (mesmas pesquisas com parâmetros diferentes).

Alguns autores também argumentaram que uma implementação deveria usar:

- Um esquema de concorrência *lock-free;*
- Uma arquitetura distribuída *shared-nothing* (onde todos os nodos são autosuficientes, sem haver ponto de contenção em toda a arquitetura) (STONEBRAKER, 2011b).

As características dos sistemas NewSQL são explanadas na Seção 2.5.1. Em linhas gerais, os bancos classificados como NewSQL têm muitos dos requisitos para gerenciamento em ambientes de computação em nuvem (ou distribuída), que se popularizam cada vez mais, e também oferecem os benefícios dos padrões já conhecidos da SQL (GROLINGER et al., 2013).

Da análise geral do trabalho pode-se dizer que o paradigma NewSQL não apresenta uma mudança radical em nenhum conceito abordado. O paradigma se apresenta mais como um apanhado das melhores características dos paradigmas anteriores. As características chave não possuem conceitos novos, mas os produtos baseados neste paradigma trazem uma implementação que traz todos esses conceitos juntos, em produtos desenhados para trabalhar numa arquitetura distribuída. Mais do que isso, mantém a compatibilidade com a linguagem de consulta SQL, o que permite que seja feita a migração de sistemas legados para trabalhar com desempenho e integridade numa arquitetura moderna. De mesma forma se mostra interessante aos sistemas novos, sendo sistemas que permitem escalabilidade horizontal e elasticidade dinâmica.

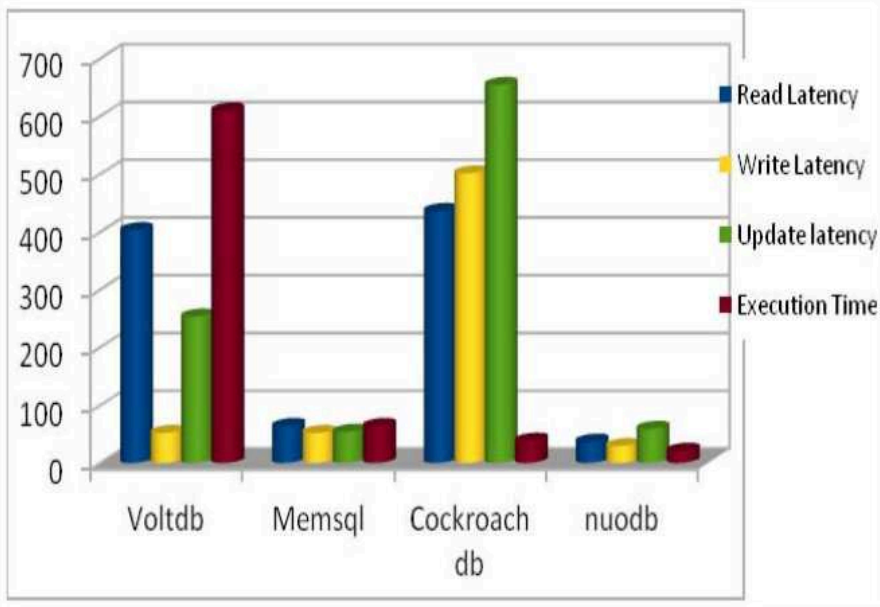
3.2 Performance evaluation of NewSQL databases

O artigo de Karambir Kaur e Monika Sachdeva (KAUR; SACHDEVA, 2017) apresenta uma avaliação de desempenho entre bancos do paradigma *NewSQL.* Os autores avaliaram 4 produtos: VoltDB, NuoDB, CockroachDB e MemSQL, as mesmas escolhas de produtos elegidos para este trabalho. Ele discorre brevemente sobre os conceitos relacionados às características que compõem o paradigma SQL, as quais são explicadas de forma mais detalhada também neste trabalho na Seção 2.5.1, e um *overview* dos produtos, também presente neste trabalho na Seção 2.5.3.

O objetivo do trabalho consistiu em estudar as soluções selecionadas e questionar se existiria suficiente conhecimento das qualidades dos sistemas NewSQL para ajudar um engenheiro de software no processo de decisão. O método de teste escolhido pelos autores analisou as operações de *create*, *insert*, *update* e *select* nos bancos de dados selecionados, para medir as características de tempo de execução, latência de leitura, latência de escrita, e latência de atualização.

Esse teste foi realizado em apenas uma instância, que mesmo com vários núcleos não representa o cenário ideal para avaliar o paradigma NewSQL. A Aplicação do teste consistiu na criação de tabelas e carga de trabalho, e a execução das mesmas. As métricas de criação, inserção, atualização e seleção foram calculadas através da saída do banco de dados. O comparativo dos resultados foi apresentado no Gráfico 1.

Gráfico 1 – Resultado da aplicação dos testes nas métricas selecionadas



http://ieeexplore.ieee.org/document/8068585/#full-text-section

O Gráfico 1 apresenta nas cores os resultados, enquanto o eixo Y apresenta o tempo decorrido em milissegundos. Os resultados obtidos mostram uma similaridade na métrica de escrita, com exceção do CockroachDB que levou mais tempo do que todos os produtos para inserir linhas na tabela. Em geral, o NuoDB se apresentou bem em todos os parâmetros, levando o tempo mínimo para executar todas as operações, enquanto o CockroachDB levou quase o tempo máximo para executar todas as operações, exceto pelo tempo de execução.

3.3 Comparative Survey of NoSQL/ NewSQL DB Systems

Nesta dissertação o autor Yuri Gurevich (GUREVICH, 2015) faz um *benchmarking* entre soluções dos paradigmas NoSQL e NewSQL. A escolha dos produtos selecionados para o teste foi baseada numa análise de tendência feita pelo autor pela fonte DB-Engines[1], fonte também utilizada por esta monografia. A análise do autor, considerou além da relevância atual na comunidade, possibilidades de consulta, existência de estrutura de *MapReduce* nos produtos, a escalabilidade, controle de concorrência, replicação, particionamento, consistência, segurança e criptografia, fazendo uma comparação das características explanadas nos produtos selecionados.

O projeto de Yuri consistiu em uma avaliação prática entre os paradigmas NoSQL e NewSQL usando para isso o *benchmark* BG[2]. O BG é um benchmarking que avalia o desempenho do armazenamento de dados processando ações de uma rede so-

[1] https://db-engines.com/en/
[2] http://www.bgbenchmark.org/BG/overview.html

cial (USC DATABASE LAB, 2015), inspirado por *benchmarks* de serviços em nuvem, como YCSB e YCSB ++, sites de e-commerce, e sistemas de processamento de transações orientados a objeto.

Para o teste com o BG, Cassandra foi o produto NoSQL usado na comparação, baseado em sua popularidade e por ser construído usando vários conceitos de dois notáveis projetos NoSQL: Dynamo[3] e BigTable[4]. Como representante do paradigma NewSQL, foi escolhido o NuoDB,por implementar internamente varias técnicas que foram citadas na fundamentação, como MVCC e replicação assíncrona.

O benchmarking BG foi escolhido pelo modelo conceitual, mais complexo que o YCSB e YCSB++ (GUREVICH, 2015). O BG se desdobra em 2 aspectos: No primeiro, dá ênfase em interações sociais que geram uma pequena quantidade de dados. Em segundo, promove uma quantidade imprevisível de dados produzidos por uma solução como a primeira métrica para comparar diferentes SGBDs. Não foram apresentadas informações quanto ao ambiente utilizado para uso do BG e dos produtos.

As métricas avaliadas foram:

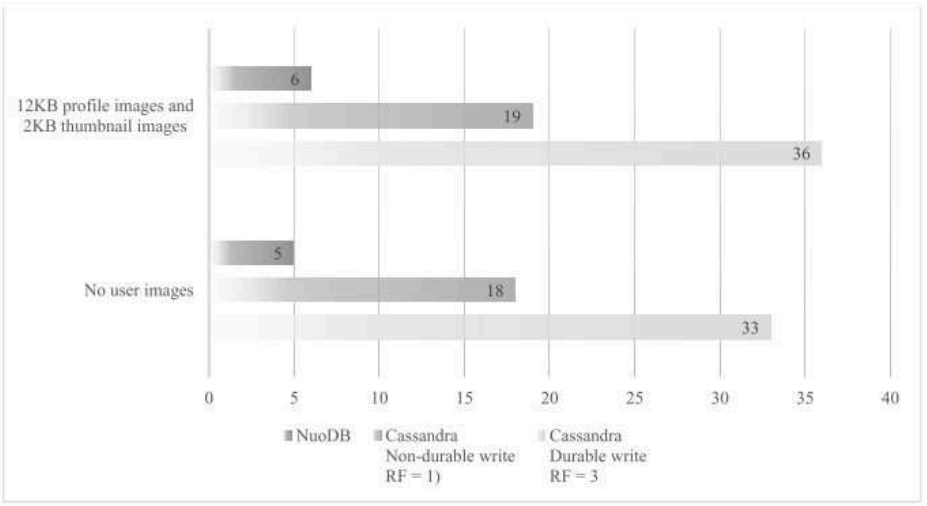
1) Tempo de população da base, que mede o tempo para o BG carregar as bases com 10 000 membros com 100 amigos por usuário e 100 recursos por usuário. O carregamento foi feito usando 10 threads;

2) *Social Action Rating* (SoAR) - Essa métrica computa o número de ações concorrentes executadas por um sistema com uma porcentagem fixa de requisições (ex. 98%) observando a latência igual ou menor do que a pré-especificada (digamos 100 milissegundos), com a soma dos dados imprevisíveis menos um limite pré-fixado (ex. 0,01%), por uma duração de tempo fixada (ex. 10 minutos). Os valores mostrados como exemplo são os parâmetros do BG. A saída do BG é a avaliação SoAR para o banco de dados alvo.

Os resultados são divididos entre as métricas avaliadas acima. Os resultados são mostrados no Gráfico 2.

[3] https://aws.amazon.com/pt/dynamodb/
[4] https://cloud.google.com/bigtable/

Gráfico 2 – Resultado do tempo de população da base para os produtos selecionados

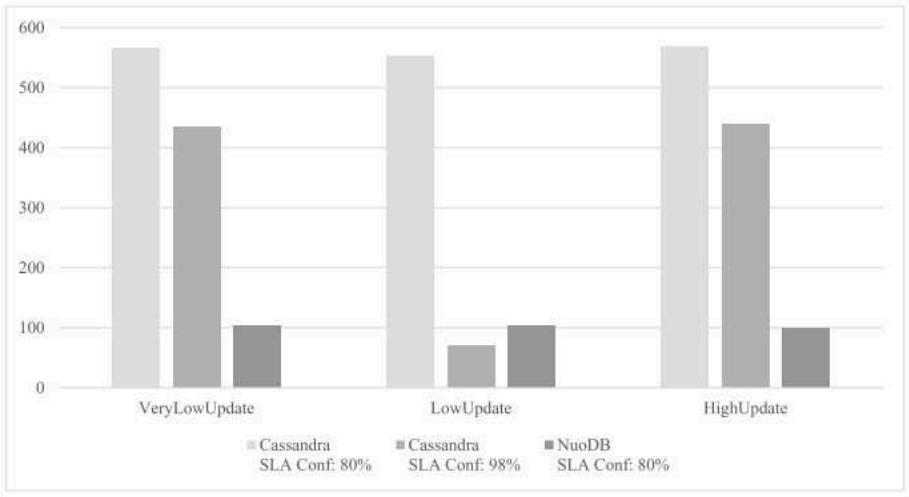


https://www.openu.ac.il/lists/mediaserver_documents/academic/cs/ComparativeSurvey.pdf

O autor destaca que bancos de dados relacionais são mais adequados ao carregamento de dados quando as relações são conhecidas e refletidas corretamente no esquema do banco. Além disso, o banco de dados Cassandra estava configurado no modo de escrita não durável (*non-durable write* - onde o fator de replicação é menor que o número de nodos), apresentando, assim, um desempenho de carregamento muito maior comparado ao modo de escrita durável (*durable write* - quando o fator de replicação é igual ao número de nodos).

Os resultados obtidos para avaliação do índice SoAR são mostrados no Gráfico 3.

Gráfico 3 – Índices SoAR dos produtos usando diferentes cargas de trabalho



https://www.openu.ac.il/lists/mediaserver_documents/academic/cs/ComparativeSurvey.pdf

De outro lado o Cassandra mostrou melhor desempenho durante o experimento

de emulação de atividades de interação social. O resultado mostrado no Gráfico 3 pode ser explicado pela descentralização das funcionalidades como a falta de normalização, que faz o banco de dados NoSQL facilmente escalável e mostra melhores resultados em uma aplicação quando múltiplos clientes executam ações concorrentes. Por último o resultado das características da arquitetura especial que contribuem para a escrita do Cassandra: Consistência eventual, falta de mecanismo de lock e uso de estruturas de memória chamados memTable.

3.4 Comparação entre os métodos utilizados

Nas seções 3.2 e 3.3 foram apresentados dois trabalhos que faziam comparações entre bancos de dados utilizando diferentes abordagens. A Tabela 1 apresenta uma comparação dos trabalhos relacionados apresentados nas seções anteriores e a proposta deste trabalho.

Tabela 1 – Comparação entre os métodos utilizados nos trabalhos correlatos

	Performance evaluation of NewSQL databases	**Comparative Survey of NoSQL/ NewSQL DB Systems**	**Análise e benchmarking das soluções NewSQL CockroachDB, MemSQL, NuoDB e VoltDB (este trabalho**
Produtos avaliados	CockroachDB, MemSQL, NuoDB e VoltDB	Cassandra e NuoDB	CockroachDB, MemSQL, NuoDB e VoltDB
Métricas avaliadas	Latência em leitura, escrita, atualização e execução	Tempo de população da base e índice SoAR	Transações por segundo(*Troughput*), Latência média do teste, Tempo do teste e métricas próprias de cada *benchmark*
Frameworks utilizados	Nenhum	BG	OLTPBench
Setup	1 instância virtual	Não informado	Ambiente Cluster 3 nodos + *Host* com aplicação OLTPBench

Na Tabela 1 são mostradas várias características dos trabalhos de modo consolidado. Nas colunas são mostrados os valores referentes aos trabalhos e nas linhas as características comparadas. Em produtos avaliados, temos para o artigo apresentado na Seção3.2 alguns representantes da categoria NewSQL, escolhidos com base na análise dos requisitos para o paradigma NewSQL. No segundo trabalho de aplicação prática na Seção 3.3, foram escolhidos um representante do paradigma NoSQL e

outro do paradigma NewSQL, objeto de estudo deste trabalho. Por fim, neste trabalho exploramos quatro bancos de dados do paradigma NewSQL.

Nas métricas avaliadas na Seção 3.2 *o*s autores utilizaram métricas simples, que podem ajudar no processo de escolha de uma solução, mas não necessariamente são as melhores métricas para se trabalhar com esse tipo de banco, visto que muitos dos benefícios do paradigma NewSQL são observados em ambiente distribuído. No segundo trabalho prático na Seção 3.3, temos o emprego de um *benchmark* na avaliação, e métricas mais robustas, como o índice SoAR, métrica oficial do *benchmark* utilizado. Entretanto, a comparação entre produtos de diferentes paradigmas tende a mascarar a avaliação das características de cada paradigma.

Por fim temos a configuração do ambiente de testes, mostrado na Tabela 1. No trabalho mostrado na Seção 3.2, foi utilizada apenas uma instância física, podendo não avaliar com clareza as características de paralelismo dos sistemas selecionados. No segundo trabalho, mostrado na Seção 3.3 não há sequer informação sobre essa configuração no texto, o que torna difícil reproduzir o experimento. Selecionar a arquitetura pensando nos produtos a se testar é importante, pois como já mencionado, esses produtos têm vantagens relacionadas a ambientes de múltiplas instâncias, então uma comparação em um ambiente com um único nodo ou configuração insuficiente pode não representar um ambiente de uso real destes produtos.

4 Implementação dos testes

Ao fazer a análise da literatura conclui-se que as comparações existentes não refletem referencial suficiente para embasamento na escolha de um banco de dados NewSQL. No trabalho mostrado na Seção 3.2, as métricas informadas consistiam em simples análises de latência, simplórias comparadas aos aspectos principais do paradigma NewSQL explanados na Seção 2.5.1, como processamento paralelo, balanceamento de carga, etc. O ambiente de testes deste trabalho também não se mostrou ideal, visto que os testes foram executados sob uma única instância física, com configuração contendo apenas 1 processador. O ideal para avaliar um sistema projetado para se beneficiar de uma arquitetura distribuída seria utilizar múltiplos nós, para garantir que os benefícios destes produtos possam ser observados.

Na análise prática mostrada na Seção 3.3 temos uma comparação entre candidatos do paradigma NoSQL e NewSQL, cenário frequentemente encontrado na revisão bibliográfica para este trabalho, mas que se mostra uma comparação não ideal sob produtos criados com diferentes perspectivas e objetivos. Um dos motivos para o surgimento do próprio paradigma NewSQL está relacionado a uma crítica ao paradigma NoSQL, para resolver problemas como a falta de *schema* e o trabalho de programação excessiva para compensar as especificidades de cada produto (PAVLO, 2016). Além disso, o trabalho correlato não considera características principais do paradigma NewSQL como o aproveitamento do processamento para cargas OLTP e OLAP.

Os produtos NewSQL selecionados para o presente trabalho foram escolhidos para este trabalho baseados nos seguintes critérios: Disponibilidade de material técnico do fabricante, existência de uma versão grátis ou de teste para execução dos testes que permitisse atingir os objetivos definidos, popularidade no ranking DB-Engines[1], mas sobretudo a adequação aos princípios do paradigma NewSQL, explanados na Seção 2.5.

O presente trabalho tem como objetivo avaliar características relevantes observadas na revisão bibliográfica e citadas na Seção 2.5.1, para os 4 produtos candidatos escolhidos, previamente apresentados na Seção 2.5.3. Os produtos foram instalados em um ambiente controlado, dividindo sua estrutura sob 3 máquinas de mesmas especificações técnicas. Sob este cluster, a ferramenta OLTP-Bench executará os *benchmarks* selecionados para o trabalho, e será criado um comparativo técnico com base nos resultados.

[1] https://db-engines.com/en/

4.1 Ferramental e *benchmarkings* utilizados na implementação

Nas subseções a seguir são apresentados conceitos referentes às tecnologias envolvidas no processo do teste proposto. São expostos conceitos sobre a ferramenta de avaliação OLTP-Bench na Seção 4.1.1, sobre os benchmarks TPC-H, YCSB e Voter nas seções 4.1.4, 4.1.2 e 4.1.3, respectivamente, e por fim sobre a tecnologia de virtualização Docker[2], explanada na Seção 4.1.5.

4.1.1 OLTP-Bench

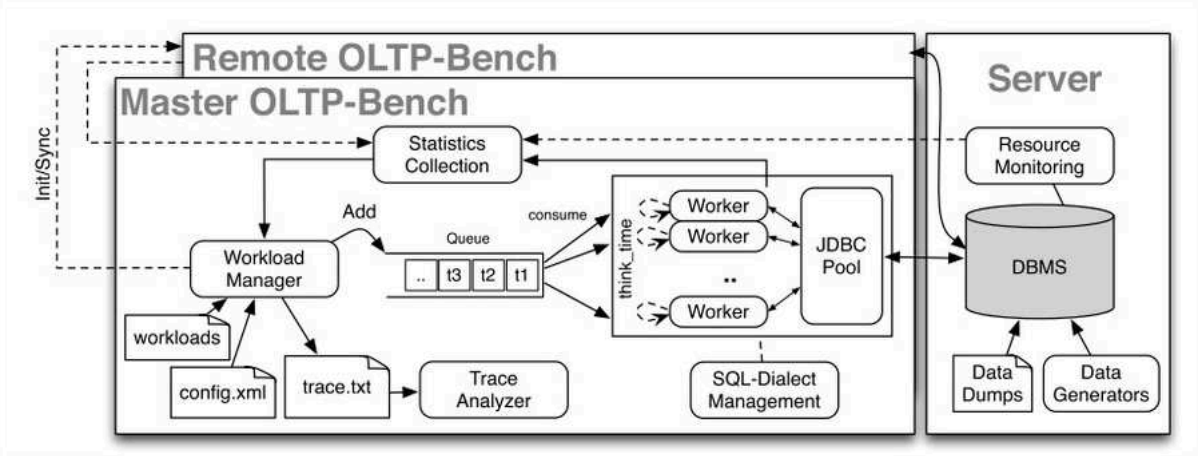
A falta de uma infraestrutura de *benchmarking* e a dificuldade de obter acesso a dados e cargas de dados reais resultam em volume de trabalho desnecessário e torna a tarefa de avaliar os resultados de uma análise de desempenho difícil de comparar (DIFALLAH et al., 2013). O OLTP-Bench (DIFALLAH et al., 2013) foi a ferramenta utilizada para avaliar os sistemas de bancos de dados neste trabalho. Ela é uma *testbed,* ou uma suíte de *benchmarking*, que supre essa demanda. A ferramenta já traz suporte a vários sistemas de bancos de dados do mercado, assim como uma boa carga de *benchmarks* diferentes, e permite expansibilidade pelo usuário para englobar novos produtos e testes, caso deste trabalho.

O OLTP-Bench apresenta fácil uso, permite extensão das suas funcionalidades, faz um bom controle das transações, avaliação de requisições, e distribuição dos acessos (DIFALLAH et al., 2013). A ferramenta mostra valor especialmente por três características: (i) Um *framework* automatizado e extensível de configurar, executar e analisar os resultados dos experimentos de desempenho do DBMS, com controladas e repetíveis configurações; (ii)Datasets reais e geradores sintéticos, com suas cargas de trabalho, todos implementados na mesma ferramenta; (iii) Descobertas experimentais sobre os DBMSs e DBaaS, resultante de centenas de experimentos.

Um dos princípios fundamentais do *design* do OLTP-Bench é que não foi imposta nenhuma regra de configuração pré-definida. Não foi pensado em restringir qualquer aplicabilidade do *framework* ou carga de trabalho para qualquer contexto, pois se acredita que é impossível definir qualquer regra que seria aplicável e relevante para todos os futuros cenários de implantação e execução. Na Figura 6 é mostrada uma visão da arquitetura do OLTP-Bench:

[2] https://www.docker.com/

Figura 6 – Visão geral da arquitetura do OLTP-Bench



http://www.vldb.org/pvldb/vol7/p277-difallah.pdf

Na parte esquerda da Figura 6 podemos ver o *driver* do lado do cliente, que lida com os *workers* e gera cargas de trabalho personalizadas, de acordo com a configuração passada pelo usuário. No lado direito, o monitor do servidor DBMS coleta as estatísticas de utilização de recursos (DIFALLAH et al., 2013). As partes mais importantes da arquitetura são mais aprofundadas a seguir.

O *Workload manager* na Figura 6 lê os parâmetros fornecidos pelo usuário , como o tamanho do pool de conexão (ou número de conexões paralelas), a composição da carga de trabalho, a vazão esperada da transação e a duração. Esses parâmetros são especificados em cada fase do experimento.

Em tempo de execução o *Workload Manager* instancia um ou mais *Worker threads (*podem ser vistos no centro da Figura 6*)* e popula a consulta de requisição que é consumida por ele. Essas *threads* também podem rodar no mesmo nodo que o *Workload Manager* ou rodar em outras máquinas.

O *OLTP-Bench* suporta 3 modelos de carga de trabalho para usar as transações: (1) *closed-loop, open-loop e semi-open-loop.* No modo *closed-loop*, OLTP-Bench inicializa um número fixo de *Workers* que repetidamente executam transações com um tempo aleatório entre cada requisição. No modo *open-loop*, a taxa em cada requisição é invocada seguindo um processo estocástico (basicamente uma família de variáveis aleatórias). Por fim, na política *semi-open-loop*, o sistema age essencialmente como um *open-loop*, com a diferença que o *Worker* pausa por um tempo aleatório antes de submeter uma nova transação. Isso simula uma carga do mundo real, onde os clientes não imediatamente submetem uma nova requisição logo que a outra foi atendida.

Em adição às funções apresentadas acima, o OLTP-Bench ainda pode lidar dinamicamente mudando o número de *Workers* e pesos das transações, para uma modelagem mais complexa. Também pode rodar vários benchmarks ao mesmo tempo,

para testar a habilidade do DBMS de balancear a carga sob diferentes circunstâncias.

O tradutor de dialeto SQL (na Figura 6 mostrado como *SQL-Dialect Management*), reescreve as transações para o formato nativo do DBMS escolhido. Além disso, o OLTP-Bench utiliza um catálogo interno universal para cada benchmark, para extrair informação sobre as tabelas e colunas do *benchmark*. Isso permite que códigos úteis do benchmark e o controle da transação sejam portados para novos DBMSs com o mínimo de esforço.

O OLTP-Bench também suporta *Workers* em múltiplas máquinas. Esse controle de instâncias é feito pelo *Workload Manager* via SSH. Ele envia o comando para iniciar e parar o *benchmark* e coleta os resultados dos nodos e faz o *merge* em um único arquivo de saída.

Por fim, para realizar a coleta de resultados em tempo de execução e a coleta de estatísticas, os *Workers* geram essas informações e o *Workload Manager* é o responsável por unir as informações sobre a execução, e a coleta de estatísticas geradas pelos *Workers* sobre o processo. Um componente chamado *Trace Analyzer* é quem faz a leitura e gera gráficos para os usuários utilizando esses dados.

Os dados de utilização no lado dos servidores são coletados por uma versão estendida de bibliotecas DSTAT[3], que captura estatística do SO (CPU, RAM, I/O, etc), e do SGBD em questão, com o mínimo de impacto no desempenho. No fim do experimento, as informações são passadas ao *Trace Analyzer* e são automaticamente alinhadas com os dados coletados.

Considerando que os produtos NewSQL são concebidos para preencher a lacuna entre o desempenho do paradigma NoSQL e, ao mesmo tempo, garantir características ACID (PAVLO, 2016), duas premissas foram observadas na escolha do *benchmark* a ser utilizado:

1) Um ou mais candidatos deveriam simular um cenário com transações mais simples, porém em grande quantidade. O cenário deve ter transações menores mais simples como *Selects, Inserts, Updates* (cargas OLTP encontradas em sistemas de operação), porém com um grande número de concorrência para testar aspectos como escalabilidade, desempenho em lidar com as concorrências, elasticidade, e disponibilidade.

2) Um candidato deveria simular um cenário como um sistema com aplicações de apoio a decisão (cargas do tipo OLAP), para o manuseio de uma grande quantidade de dados e observar o desempenho da leitura intensiva de dados através de consultas simples e complexas;

[3] http://dag.wiee.rs/home-made/dstat/

Na avaliação dos candidatos a *benchmark*, foram selecionados três candidatos que supriam as necessidades apresentadas. Os candidatos selecionados para lidar com cargas OLTP e as características definidas na primeira premissa apresentada foram o YCSB e o Voter. O candidato escolhido para lidar com as cargas OLAP foi o TPC-H. Esses *benchmarks* são suportados pela ferramenta OLTP-Bench. Mais informações sobre os candidatos são explanadas a seguir.

4.1.2 YCSB

O Yahoo! *Cloud Serving Benchmark* (YCSB) foi desenhado para ser não somente um *benchmark*, mas um *framework* de *benchmark* para auxiliar na avaliação de diferentes sistemas em nuvem, focando em sistemas que têm forte interação com o usuário (carregamento de página, leituras e escritas de dados de uma base de dados, etc), mas são disponibilizados em um serviço online (COOPER et al., 2010).

O framework YCSB consiste em uma aplicação geradora de carga de trabalho e um pacote com cargas de trabalho padrão que cobrem interessantes partes da avaliação de desempenho (cargas de trabalho de leitura intensa, cargas de trabalho de escrita intensa, varredura de tabelas, etc.). O ponto principal citado pelos autores do artigo é a extensibilidade da ferramenta, onde o gerador de cargas foi desenhado de forma a permitir ao desenvolvedor criar tipo de carga de trabalho, e adaptar o *benchmark* a novos produtos que podem surgir (COOPER et al., 2010).

Os conjuntos de cargas de trabalho usadas no YCSB recebem o nome de YCSB *Core Package*. Cada carga de trabalho representa um misto de operações de leituras e escritas, usando diferentes volumes de dados e números de requisições. Os usuários do YCSB podem desenvolver uma carga de trabalho personalizada para seus testes, ou utilizar as cargas já desenvolvidas que estão sendo distribuídas em código aberto. As cargas são desenvolvidas com o objetivo de avaliar várias características do banco de dados, conseguindo abranger diferentes tipos de bancos de dados. Por exemplo, alguns sistemas são otimizados para leituras, mas não para escritas, ou para inserções, mas não atualizações, etc. Estas características distintas e seus *trafeoffs* têm de ser considerados, e o YCSB lida com esse tipo de nuances.

A estrutura deste teste consiste em uma única tabela nomeada como *usertable*, com N campos. Cada registro é identificado como uma chave primária, algo como “user234123”. Cada campo é nomeado como field0, field1, e assim por diante. Os valores dos campos são *strings* randômicas de caracteres ASCII de tamanho aleatório. Os parâmetros de número de campos(N) e o fator de escala (F) são informados, por isso não constam como número fixo. Cada operação sobre esta base de dados corresponde a um dos tipos abaixo mostrados:

- Atualização (*Update*) - Atualiza um registro, substituindo o valor de um dos campos;
- Leitura (*Read*) - Lê um registro, de um campo aleatoriamente escolhido entre os disponíveis na tabela;
- Seleção (*Scan*) - Examina registros de uma tabela em ordem, iniciando de um registro aleatório. O número de registros a examinar também é escolhido aleatoriamente.

Para o *Scan* especificamente, a distribuição do tamanho desta operação é escolhida como parte da carga de trabalho. Num sistema real geralmente se especificaria um intervalo (ex: de 1 de fevereiro até 15 de fevereiro). O parâmetro número de registros nos permite controlar o tamanho desses intervalos, sem precisar determinar e especificar pontos de extremidade significativos para a varredura.

A execução do teste realiza muitas escolhas aleatórias gerando a carga do teste: As operações que serão feitas (dentre *Insert*, *Update*, *Read* or *Scan*), qual registro ler ou escrever, quantos registros examinar, e assim por diante. Essas decisões são governadas por distribuições randômicas. As distribuições utilizadas pelo YCSB são:

- Uniforme (*Uniform*) - Escolhe um item uniformemente, de forma aleatória. Uniformemente quer dizer que todos os registros têm possibilidades iguais de serem escolhidos;
- Distribuição de Zipf (*Zipfian*) - Escolhe um registro de acordo com a distribuição abordada pela Lei de Zipf[4]. A lista de registros é ordenada pela frequência de aparição, iniciando pelos mais populares e terminando com os que menos aparecem. Nessa distribuição não se considera a ordem de inserção dos registros.
- Recentes (*Latest*) - Utiliza uma estrutura muito semelhante ao *Latest*, exceto pelo fato de que os registros inseridos por último são colocados como primeiros na ordenação. É mais usada em aplicações onde o momento de inserção do registro importa, tendo como um bom exemplo o conteúdo postado no site do Twitter(o *tweet),* onde um assunto se torna relevante conforme surgem, num tempo semelhante, menções a ele em dado momento.
- Multinomial - As probabilidades de cada item podem ser especificadas. Por exemplo, podemos definir a probabilidade de 95% de ser uma operação *Read*, 5% de probabilidade de ocorrer um *Insert*, deixando 0% para *Selects* e *Inserts*, formando uma pesada carga de leitura.

[4] https://pt.wikipedia.org/wiki/Lei_de_Zipf

O benchmark YCSB cria 5 cargas de trabalho, com variações entre as operações que realização, dispostas na Tabela 2.

Tabela 2 – Informações sobre as cargas de trabalho disponíveis no benchmark YCSB

Carga de trabalho	Distribuição de operações	Método de distribuição adotado	Exemplo de aplicação
A - Atualizações intensas	*Read*: 50%, *Update* 50%	Distribuição de Zipf	Gravação de ações recentes de um usuário numa sessão
B- Leituras intensas	*Read* 95%, *Update* 5%	Distribuição de Zipf	Etiquetação de fotos: Adicionar etiquetas é uma atualização, mas a maioria das operações são de leituras das etiquetas existentes.
C- Leituras apenas	*Read* 100%	Distribuição de Zipf	*Cache* de perfil de usuário, onde os perfis são construídos em outro lugar (ex: Hadoop)
D- Leituras recentes	*Read*: 95%, *Insert*: 5%	Distribuição de Zipf	Atualização de *status* de usuários: Onde os usuários querem ler as atualizações mais recentes
E- Alcances pequenos*	*Scan* 95%, *Insert* 5%	Distribuição de Zipf/ uniforme*	Conversações individuais, onde cada operação de *scan* é par ler os *posts* mais recentes em cada conversa

COOPER, B. F. et al.(2010)

*A carga de trabalho E utiliza a distribuição de *Zipf* para escolher a primeira chave no intervalo, e a distribuição uniforme para escolher o número de registros a serem examinados.

Mostradas na Tabela 2 temos as 5 cargas disponíveis no YCSB nas linhas, e suas informações segmentadas nas colunas. As combinações de operações mostradas na segunda coluna da Tabela 2 mostra o esforço deste *benchmark* em tentar abranger sistemas de bancos de dados que priorizam diferentes tipos de dados ou operações.

4.1.2.1 Parâmetros definidos para o *benchmark* YCSB

Nos parâmetros definidos para o *benchmark* YCSB, configuramos fatores relacionados à escala de volume da base de teste, e os dados da carga de trabalho. Os parâmetros são relacionados abaixo, junto a uma breve introdução e os valores considerados para este trabalho.

- <scalefactor> - É o fator de escala de volume de dados na tabela. O YCSB, como mencionado na seção 4.1.2, possui apenas uma tabela. A cada 1 unidade

do fator de escala, são inseridas 1000 tuplas na tabela USERTABLE. Neste trabalho foram feitos testes com três cenários de fator de escala: 10, 100 e 1000.

- <terminals> - Se refere ao número de usuários virtuais simultâneos que serão utilizados para manipular o banco do YCSB. Neste teste, utilizamos 64 usuários virtuais.

- <weights> - É o parâmetro que define a proporção de execução de cada tipo de carga[5], em porcentagem. Recebe 6 valores, referentes aos seguintes tipos de carga: (i) atualizações (updateproportion); (ii) escaneamentos (leitura total de tabela) (*scanproportion*); (iii) leituras (*readproportion*); (ii) atualizações (updateproportion); (iii) inserções (*insertproportion*); (iv) uma operação que envolve a leitura do registro, atualiza o valor e escreve no banco de dados (readmodifywriteproportion); inserção(*insertproportion*) e (vi) deleções (*deleteproportion*) . Neste trabalho, dedicamos mais valor ás operações de leitura considerando o contexto do teste, então o valor setado foi "50,5,15,10,10,10" ou seja, 50%, 5%, 15%, 10%, 10% e 10% dos 6 tipos de pesos explanados.

- <time> - tempo de execução máxima do teste, em segundos. Neste valor, como não se trabalha com fatores muito grandes, foi setado o limite de 300 segundos, valor "300";

- <rate> - se refere ao número de operações que o teste irá performar, em unidades. O valor aqui é setado como sem limites, valor "unlimited", pois a única limitação deve ser o tempo de 300 segundos.

4.1.3 Voter

O Voter é um *benchmark* baseado em um software utilizado em um programa de talentos exibido em televisão, nos países do Japão e Canadá. Os usuários ligam para votar no seu candidato favorito.

Ao receber a ligação a aplicação invoca a transação que atualiza o número total de votos de cada participante. O sistema de banco de dados então registra o número de votos feitos por cada usuário, fixado em um limite máximo. Uma transação em separado é periodicamente invocada para computar os votos totais durante o programa.

Este benchmark foi desenhado para saturar o banco de dados com pequenas transações, todas atualizando um pequeno número de registros. A arquitetura deste teste se dá através de três tabelas que guardam algumas informações sobre os

[5] https://github.com/brianfrankcooper/YCSB/wiki/Core-Properties

candidatos e o usuário que está ligando, e duas *Views*, que são consultadas para atualizar o *status* no programa de televisão.

4.1.3.1 Parâmetros definidos para o *benchmark* Voter

Nos parâmetros definidos para o *benchmark* Voter configuramos fatores relacionados à escala de volume da base de teste, e os dados da carga de trabalho. Os parâmetros alterados são relacionados abaixo, junto a uma breve introdução e os valores considerados para este trabalho.

- <scalefactor> - É o fator de escala de volume de dados na tabela. O Voter, como mencionado na seção 4.1.2, possui um conjunto de 3 tabelas e 2 *views.* O fator de escala aumenta o volume das tabelas, e consequentemente torna o carregamento das *views* maior. O fator de escala utilizado no trabalho foi 10.
- <terminals> - Se refere ao número de usuários virtuais simultâneos que serão utilizados para manipular o banco do Voter. Neste teste, utilizamos 30 usuários virtuais.

4.1.4 TPC-H

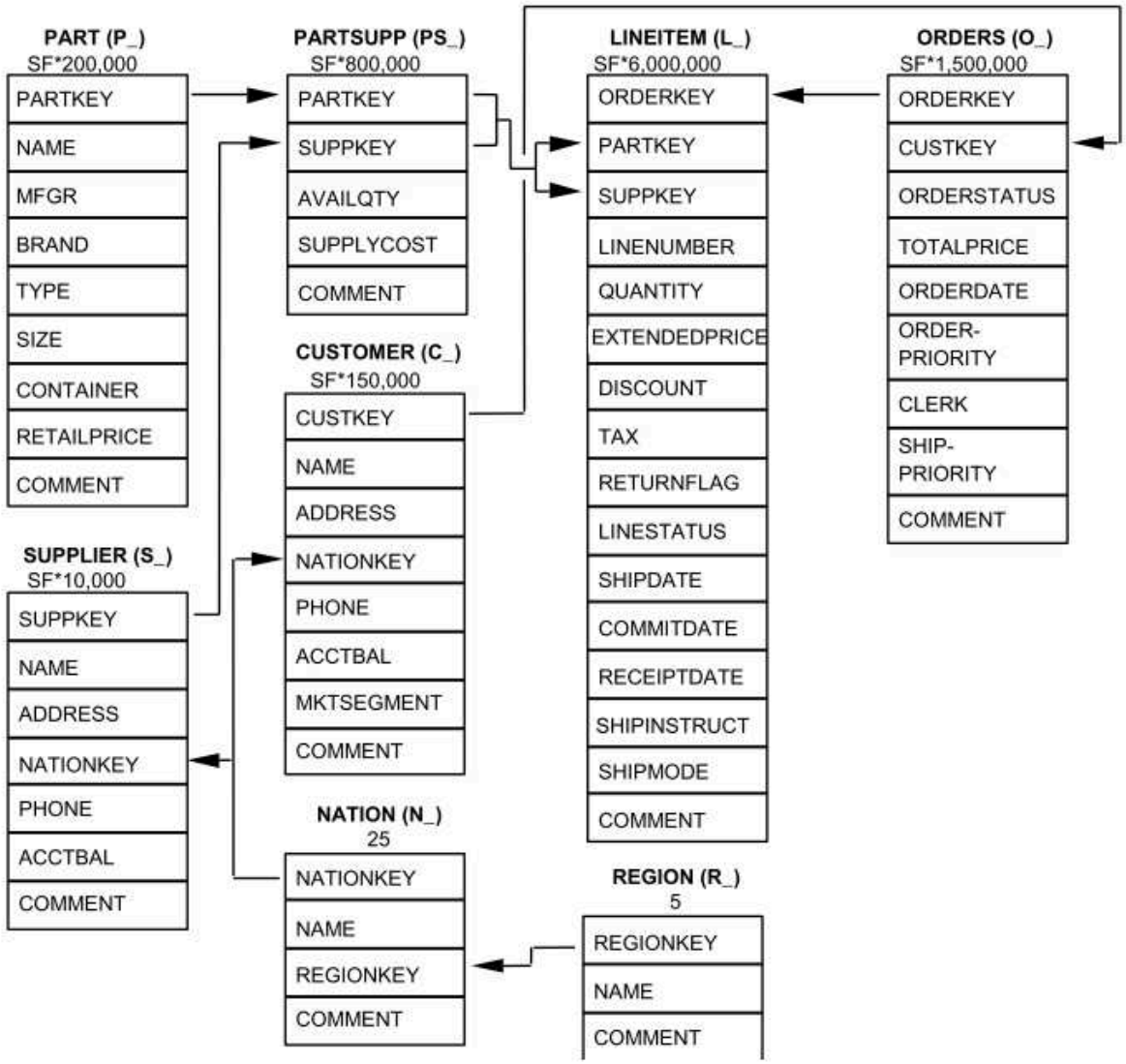
De acordo com a documentação oficial do TPC-H (TRANSACTION PROCESSING PERFORMANCE COUNCIL (TPC), 2017) este *benchmark* apresenta uma suíte de *queries ad-hoc* empresariais e modificações de dados concorrentes. O TPC-H avalia o desempenho de vários sistemas de apoio a decisão pela execução de um conjunto de consultas sob uma base em condições controladas. As consultas do TPC-H tem como características: (i) Dar resposta a questões empresariais do mundo real; (ii) Simular consultas *ad-hoc* aleatórias; (iii) ter uma complexidade maior do que a maioria das consultas de cargas OLTP; (iv) Gerar atividade intensa por parte do servidor; (v) uma implementação com restrições alinhadas com uma base de produção *online.*

O cenário do TPC-H consiste em um ambiente de negócios onde transações são executadas em tempo real. Neste cenário, são executadas atualizações periódicas sob uma base de suporte a decisão (num modelo de *Data Warehouse*), o qual é consultado por vários tomadores de decisão concorrentemente, tudo isso conectado a uma base OLTP, funcionando online ao mesmo tempo.

Sobre a arquitetura da base de dados, o esquema representa arquivos de um importador multinacional fictício e revendedor de partes e suprimentos industriais. Os clientes e fornecedores deste negócio vem de diferentes partes do mundo (*regions*) e diferentes países (*nations*) com estas regiões. Os clientes fazem diversos pedidos e cada pedido contém muitas partes diferentes compradas de diferentes fornecedores

a diferentes preços. A lista das partes e a lista de fornecedores são conectados por uma tabela (*partsup*) para indicar as partes específicas fornecidas por cada fornecedor(BARATA; BERNARDINO; FURTADO, 2014). A representação do esquema das tabelas é mostrada na Figura 7:

Figura 7 – emphSchema do *benchmark* TPC-H



http://www.tpc.org/TPC_Documents_Current_Versions/pdf/tpc-h_v2.17.3.pdf

Na Figura 7, os parênteses em cada nome de coluna apresentam o prefixo do nome das colunas em cada tabela. As setas mostradas ligando as tabelas mostram uma relação de um-para-muitos entre as tabelas, e por fim os números apresentados acima de cada tabela representam a cardinalidade (número de linhas) de cada tabela. Algumas tabelas apresentam as letas SF, que significa o fator de escala (*Scale Factor*). Esse fator é um número informado como parâmetro ao gerar as cargas para as tabelas no teste, que determina o multiplicador tamanho das tabelas.

Considerando o fator mínimo de escala do TPC-H, a carga engloba informações

de 10 000 fornecedores, com quase dez milhões de linhas representando a carga total, somando cerca de 1GB de espaço, dividido entre as 8 tabelas de sua estrutura (TRANSACTION PROCESSING PERFORMANCE COUNCIL (TPC), 2017). Nessa estrutura são executadas 22 consultas de leitura exclusiva (*read-only*) e 2 consultas de atualização de dados. Essas consultas são executadas sob grandes quantidades de dados, com algum nível de complexidade, para responder a questões criticas de negócio.

Em sua execução o TPC-H possui duas fases de teste: A de carregamento, que mede a população das tabelas, e a de desempenho , que mede o desempenho do sistema no manuseio de uma carga específica. Ao concluir a fase de carregamento, a fase de desempenho que roda duas vezes, inicia. No teste de desempenho ocorre um teste de força, que mede a execução de uma *query* com apenas uma sessão ativa, rodando as 22 consultas do teste em seguida; um teste de vazão, onde o benchmark identifica como processar a maioria das consultas em menos tempo possível em um ambiente multi usuário e por fim, é medida a quantidade de dados que é transferida em um período de tempo. Ao terminar estes testes, intervalos são definidos para executar as funções de atualização dos dados para o sistema de decisão, para depois produzir as métricas de negócio (BARATA; BERNARDINO; FURTADO, 2014).

4.1.4.1 Parâmetros definidos para o *benchmark* TPC-H

Nos parâmetros definidos para o *benchmark* TPC-H configuramos fatores relacionados à escala de volume da base de teste, e os dados da carga de trabalho. Os parâmetros são relacionados abaixo, junto a uma breve introdução e os valores considerados para este trabalho.

O *benchmark* TPC-H não gera sua carga de teste em execução. A geração da carga de teste é feita externamente, por uma aplicação chamada DBGen[6]. Esta aplicação gera arquivos com os valores que serão inseridos como tuplas no *schema* do *benchmark*, mostrado na seção 4.1.4. Nesta aplicação, definimos a escala de tamanho das tabelas. Para cada unidade, o volume de algumas tabelas é multiplicado pelo fator de escala. Para mais informações, consulte a seção 4.1.4. A escala utilizada para geração de carga neste teste foi 1. Mesmo sendo apenas 1 unidade, o volume de registros já é suficiente para avaliar os aspectos necessários.

Outros parâmetros passados ao OLTP-Bench para execução do benchmark TPC-H estão descritos a seguir.

- <datadir> - Após geração pelo DBGen são gerados arquivos de extensão tbl (.tbl). Estes arquivos podem ser movidos para uma pasta do sistema, e o caminho para esta pasta deve ser especificado neste parâmetro;

[6] https://github.com/electrum/tpch-dbgen

- <fileFormat> - Tipo do arquivo que contém as informações das tuplas a serem carregadas, no nosso caso tbl, como mencionado anteriormente;
- <terminals> - Número de usuários virtuais que serão emulados pelo teste. No nosso caso, o valor é 1, ou seja, um usuário executará as tarefas, emulando uma tarefa programada que faz o carregamento dos dados em determinado momento definido, do banco operacional para o analítico;

4.1.5 Docker

O Docker[7] é uma plataforma aberta para desenvolver e executar aplicações. Essa tecnologia permite encapsular as dependências e os arquivos da aplicação em um ambiente isolado. A este ambiente isolado se dá o nome de contêiner (*container)*.

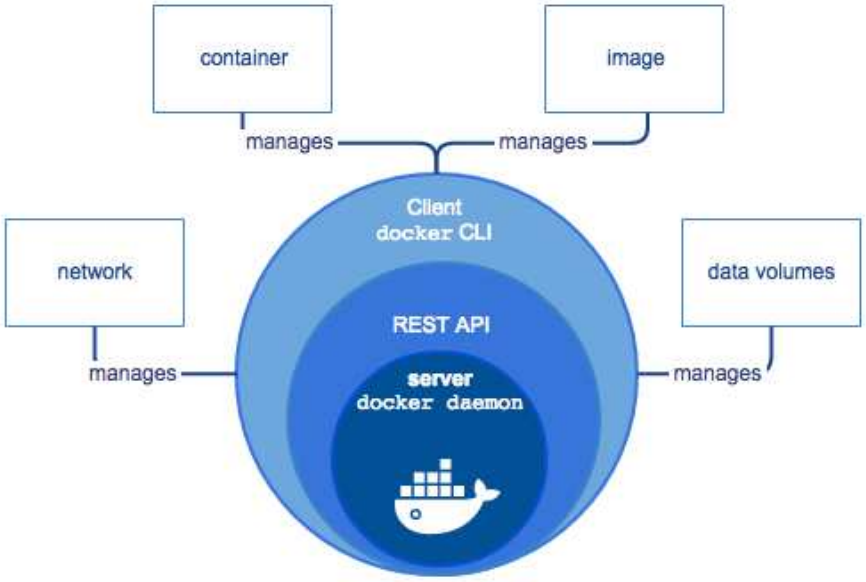
Este isolamento permite que sejam executadas várias aplicações simultaneamente, sem afetar as configurações da máquina hospedeira(*host*), então é comum confundir esta tecnologia com tecnologias de máquinas virtuais, como VirtualBox[8] e VMWare[9]. Um contêiner Docker roda nativamente nos sistemas Linux, e divide o *Kernel* do sistema com outros contêineres. Ele roda um processo discreto no sistema, que consome uma quantidade muito pequena de memória, o que o torna muito leve. Em contraste, uma máquina virtual tradicional roda um sistema operacional completo sob a camada *hypervisor* (a camada que fornece e gerencia o acesso da máquina virtual aos recursos físicos da máquina hospedeira), o que se mostra geralmente oneroso para testar aplicações, por exemplo.

Para auxiliar no isolamento da aplicação o Docker possui componentes (chamados de objetos) que dão suporte às suas funções. Entre esses objetos, os mais utilizados são: (i) imagens (*images*) - que são os arquivos que contém os arquivos que serão utilizados na execução de um contêiner; (ii) redes internas (*networks*) - que são sub-redes criadas pelo serviço para que os contêineres possam se identificar e trocar informações entre si, independente da rede de dados do hospedeiro; e (iii) volumes, que são os diretórios em que são guardados arquivos na execução dos contêineres e/ou o mapeamento de recursos do hospedeiro para dentro do contêiner Docker.

A arquitetura do serviço Docker é baseada em três componentes principais, mostrados na Figura 8 e explanados a seguir.

[7] https://www.docker.com/
[8] https://www.virtualbox.org/
[9] https://www.vmware.com/br.html

Figura 8 – Arquitetura da aplicação Docker



https://docs.docker.com/engine/docker-overview/#docker-engine

Da camada mais interior para a mais exterior do círculo na Figura 8, temos o serviço que serve a aplicação (*docker daemon*). Ele cria e gerencia os objetos Docker previamente mencionados. A camada superior (*REST API*) é responsável pela interface entre os programas e o serviço principal (*docker daemon*), e instruir as tarefas para execução. Na última camada temos a *interface* cliente (*Client docker CLI*). Essa camada utiliza a *API REST* para controlar ou interagir com o processo *docker daemon*, gerenciando os recursos dos contêineres em execução, e gerenciando aspectos da execução destes.

4.2 Arquitetura física

Nesta seção são detalhadas as características físicas do cluster utilizado, e uma visão geral sobre a implementação dos produtos nesta arquitetura.

4.2.1 Implementação do ambiente

Para implementar o conjunto de testes foi utilizado, em condições controladas, um cluster local com 3 nodos de configuração idêntica, que serão os nós do cluster. A configuração destes nodos é mostrada na Tabela 3 a seguir.

Além destes três nodos, um quarto computador foi utilizado para execução do software OLTP-Bench de maneira independente. Os comandos foram enviados deste

Tabela 3 – Configuração dos nodos do cluster

Tipo	Descrição
Software	• Sistema operacional XUbuntu 16.04 Server LTS 64-bits;
Hardware	• CPU: Intel Core Processador Intel® Core™ i5-7200, com 4 núcleos físicos de 2,50 GHz ; • Memória RAM 8GB DDR3 1333Mhz; • Disco rígido de 320GB, operando na velocidade de 5400RPM; • Conexão local de 100 Mbps entre os nodos.

O autor(2018)

computador para a API dos produtos avaliados, via conexão local com o cluster. A configuração deste computador é mostrada na Tabela 4.

Tabela 4 – Configuração do nodo que executou o OLTP-Bench

Tipo	Descrição
Software	• Sistema operacional Ubuntu 18.04 Server LTS 64-bits;
Hardware	• CPU: Intel Core Processador Intel® Core™ i7-6500U, com 4 núcleos físicos de *clock* real de 2,50 GHz ; • Memória RAM 12GB DDR3 1666Mhz; • Disco de estado sólido de 250GB; • Conexão local de 100 Mbps entre os nodos.

O autor(2018)

4.2.2 Implementação dos produtos

Utilizando o cluster local mencionado na seção anterior, foi realizada a instalação dos produtos para teste. As instalações buscaram seguir as configurações padrão dos produtos, não sendo feita nenhuma otimização, de modo a não favorecer nenhum dos resultados.

O produto VoltDB foi instalado utilizando contêineres *Docker*, estrutura previamente mencionada na Seção 4.1.5 deste trabalho. Os arquivos foram disponibilizados pela própria fabricante, no repositório oficial do *Docker*[10]. A estrutura do VoltDB necessita que um cluster ao menos seja o mestre, então um dos nodos aleatoriamente foi eleito como mestre do *cluster*. Os outros utilizaram do endereço do mestre para se unir ao cluster. Mais informação sobre a instalação do VoltDB podem ser encontradas no Apêndice A.

Os produtos CockroachDB, MemSQL e NuoDB foram instalados de forma tradicional, seguindo as orientações dos fabricantes para instalação[11,12,13]. Sobre as nuances de cada produto, o CockroachDB apresentou a instalação mais fácil entre os produtos, sendo necessários poucos e simples comandos para colocá-lo *online*. Os nós se comportam de forma síncrona, então as requisições para a API do CockroachDB podem ser feitas para qualquer um dos nodos. Mais detalhes sobre essa instalação podem ser encontrados no Apêndice B.

O produto MemSQL traz automatizações que facilitam a configuração do cluster. É necessário apenas instalar o produto em um dos nodos, e acessar a interface gráfica, que questionará se a instalação é num ambiente distribuído, e caso sim, o endereço dos nós folha. Fornecidos esses dados, ele se encarrega de fazer uma conexão SSH com os nós informados, e faz a instalação e agregação desses nós ao cluster. Mais informações sobre essa instalação pode ser obtida no Apêndice C.

O NuoDB também funciona de forma automatizada no gerenciamento do cluster. Na instalação um mestre necessita ser eleito, e na instalação dos produtos em outros nodos há uma configuração que aponta os nós para o nó mestre eleito. A versão gratuita do produto é a única entre os produtos que não fornece uma interface gráfica para o monitoramento do cluster. A estrutura do NuoDB, explanada na Seção 2.5.3.2, também tem limitações na versão gratuita, onde foi possível somente utilizar o armazenamento de dados em um nó do cluster, e os outros nós hospedaram gerenciadores de transação para dar velocidade ás consultas. Na criação da base de dados, o próprio gerenciador do NuoDB se encarrega de criar os processos gerenciadores de transação e processos

[10] https://hub.docker.com/r/voltdb/voltdb-community/
[11] https://www.memsql.com/install/
[12] https://www.cockroachlabs.com/docs/stable/install-cockroachdb.html
[13] https://www.nuodb.com/dev-center/community-edition-download

gerenciadores de armazenamento nos nós do cluster. Assim como o CockroachDB, a interface com a API pode ser feita de qualquer nodo do cluster. Mais informações sobre essa instalação podem ser obtidas no Apêndice D.

4.2.3 Métricas avaliadas

Métricas são medidas quantitativas de modo a validar em várias dimensões o desempenho de sistemas. Neste trabalho as métricas avaliadas focaram em aspectos considerados relevantes em um sistema produzido, baseados no conjunto de dados que o *framework* OLTP-Bench fornece sobre cada um dos testes selecionados nas seções anteriores.

Das métricas levantadas, optou-se por avaliar primariamente duas métricas: A taxa média de transações executadas por segundo (*Throughput*), e a latência das transações, através da análise da média geral, as latências específicas de cada teste (por tipo de transação no YCSB e por consulta no TPCH) e análise da média dos percentis. Adicionalmente foi realizado o cálulo do desvio padrão para permitir ao leitor uma melhor compreensão dos resultados. As métricas são detalhadas a seguir.

A análise da taxa de *Throughput* representa a quantidade de transações por unidade de tempo, no nosso caso, segundos. Essa métrica tem o objetivo de avaliar a capacidade de executar várias operações em simultâneo (paralelismo) no sistema (CURINO et al., 2012).

A segunda medida avaliada nos testes realizados é a latência. A medida também recebe o nome de tempo de resposta, uma vez que representa o tempo que uma operação leva para se completar. Ela pode ser considerada uma medida interessante quando aplicada a sistemas distribuídos, considerando que a garantia de propriedades e os benefícios do banco de dados distribuídos depende, em grande parte, da comunicação entre os nodos.

Há diversas formas de se avaliar latência, sendo a mais comum a análise da média. Incluímos a análise da média de forma a fazer comparação com a análises de latência específicas dos *benchmarks* YCSB e TPCH, por tipo de transação ou separado por transação, respectivamente.

Considerando um conjunto de cargas diversas, como é o caso do presente trabalho, uma pequena parte das transações pode representar uma grande parte da latência do teste todo. Para avaliar este aspecto, utiliza-se a análise de percentis.

Basicamente, na análise de percentis, a curva normal gerada é dividida em 100 partes de percentagem de dados aproximadamente igual. Assim, é possível observar se certa parte da curva foi responsável por uma maior ou menor parte do resultado do teste. O valor da latência observada no 99^o percentil, portanto, corresponde ao tempo

de resposta que 99% das operações levaram. Este trabalho emprega a análise das médias de percentis 90 e 99 obtidos para as transações.

A análise do desvio padrão agrega informação da uniformidade dos resultados nas outras métricas coletadas.

5 Análise dos resultados

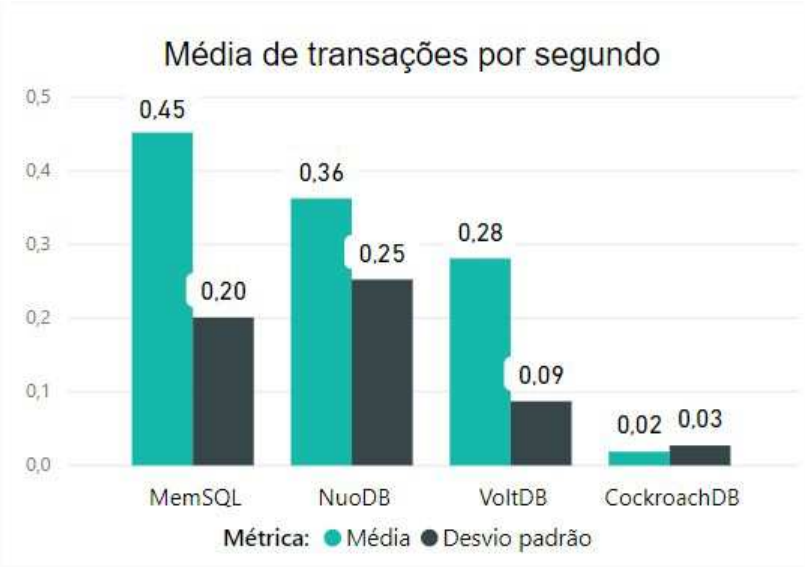
Neste capítulo são apresentados os resultados dos testes previamente descritos nas Seções 4.1.2 , 4.1.3 e 4.1.4. De modo a avaliar características pertinentes a sistemas distribuídos, avaliamos em todos os *benchmarks,* as médias de transações executadas por segundo (*Throughput*), e as análises de latência, medidas previamente descritas na Seção 4.2.3. Também foram avaliadas medidas de latência específicas de cada teste.

5.1 Resultados para o *benchmark* YCSB

Na avaliação do *benchmark* YCSB utilizamos os parâmetros mencionados na Seção 4.1.2.1. Utilizamos de três cenários para observar os resultados em um teste limitado a 300 segundos, onde a diferença foi o fator YCSB, que aumenta o volume da tabela exponencialmente. Contudo, as métricas avaliadas consideram as médias. Assim sendo, os resultados foram muito semelhantes, o que mostra que os sistemas possuem um alto nível de escalabidade linear. Como não houve variações entre as médias, optou-se por exibir apenas os valores obtidos para o valor mais alto de parâmetro, com o fator de escala do YSCB setado em 1000. O resultado foi coletado com dados de 5 execuções deste teste para cada produto.

No Gráfico 4 analisamos a primeira métrica, do número de transações executadas por segundo ao decorrer do teste.

Gráfico 4 – Média de transações por segundo obtida no *benchmark* YCSB ao decorrer do teste para os produtos.

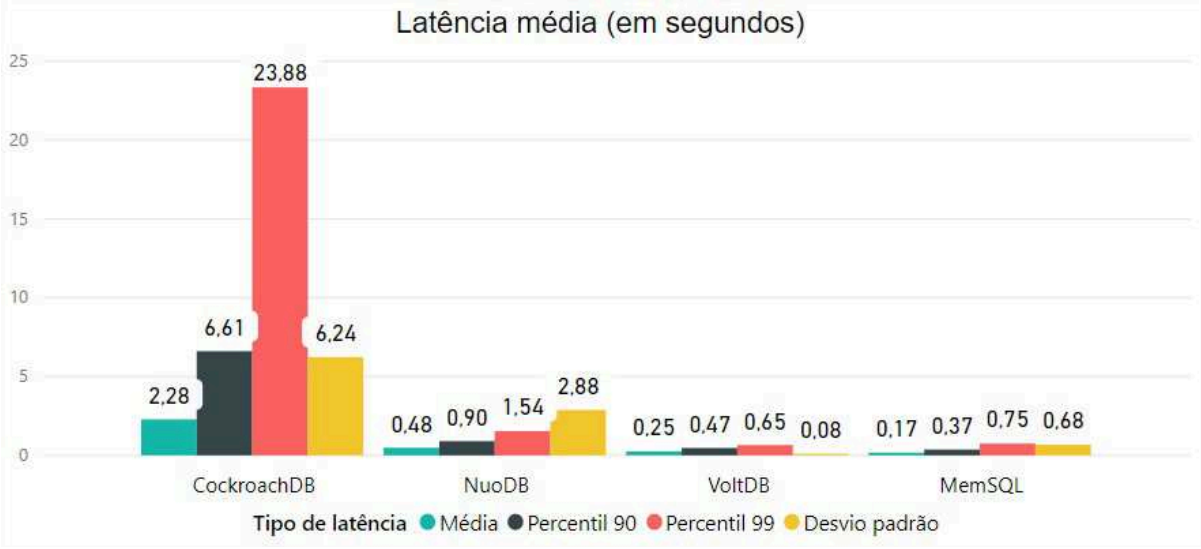


O autor (2018).

Como podemos avaliar pela análise do gráfico, o MemSQL apresentou melhores resultados na categoria executando em média 0,45 transações a cada segundo do teste, a frente do NuoDB, que executou 0,36. VoltDB por sua vez executou 0,28 transações por segundo em média, e por último temos o Cockroach, executando 0,02 transações por segundo em média. O desvio padrão mostra que o grau de dispersão entre os resultados foi baixo, mostrando uma boa uniformidade na execução, sendo um puco mais discrepante na execução do MemSQL e NuoDB.

A diferença considerável obtida no Gráfico 4 do produto CockroachDB para os concorrentes pode ser explanada comparando as demais métricas. Outro fator importante para a compreensão do resultado é a média de latência das transações executadas. O resultado de análise da latência média é mostrado no Gráfico 5.

Gráfico 5 – Latências médias obtidas no *benchmark* YCSB ao decorrer do teste para os produtos



O autor(2018)

Como se pode observar no Gráfico 5, o CockroachDB também apresenta maior latência média nas transações, quando não se considera o tipo de transação. O desvio padrão mostra que o grau de dispersão entre os resultados foi alto nos casos do CockroachDB e NuoDB, mostrando que as observações dos resultados destes produtos tiveram uma grande discrepância de valores, diferente dos outros produtos que mostraram uma execução mais uniforme do teste. Em relação a média, NuoDB e VoltDB sucedem o CockroachDB no ranking, mostrando resultados parecidos entre si. A melhor latência média fica por conta do MemSQL, com uma latência de 0,17 por segundo.

A análise dos percentis 90 e 99 mostra que quantidade expressiva das transações com maior latência se encontram em um pequeno número de transações situadas nesses pontos, representando mais que o dobro da média situada no percentil 99. O

caso é mais evidente no uso do CockroachDB, onde temos uma grande discrepância entre a média de 2,28 segundos na média de latência, e 23,88 segundos na média de latências no percentil 99.

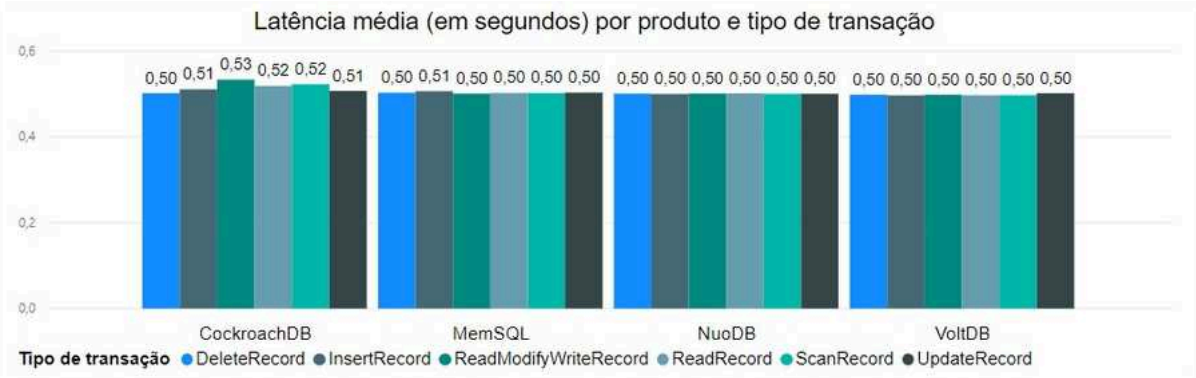
A análise da discrepância observada pelo produto CockroachDB em latência média e média de transações por segundo apresentadas deve considerar que o armazenamento principal do CockroachDB não é em memória principal (RAM) como os outros, e sim uma arquitetura *key-value* em disco que é explanada na Seção 2.5.3.3 e reforçada na Tabela 2.5.3.6. Este banco de dados utiliza um percentual da memória principal (RAM) para *cache*, mas utiliza grande parte de suas operações baseadas em disco, o que gerou as latências e taxa de vazão observados.

As discrepâncias também são notadas no produto NuoDB. Tanto na média de transações por segundo, quanto na análise de latência média, o produto mostra um grau de dispersão um pouco elevado entre os resultados. Tais resultados tendem a ocorrer tendo em vista as limitações da versão gratuita, onde é possível utilizar o armazenamento em apenas um dos nodos do cluster, mesmo que seja possível utilizar o componente de execução de transações em três nodos. O armazenamento no *storage* único pode aumentar a latência da execução, visto que os nodos executam parte do teste em um nodo, transferindo todo o resultado para o nodo que salva os dados.

Por sua vez, o VoltDB ocupa o terceiro lugar no teste. O VoltDB, produto escrito em Java, pode em sua execução ter influência do *garbage colector* da própria *Java Virtual Machine* (JVM), o que deve ser um ponto a considerar nos resultados dos benchmarks avaliados neste trabalho. Mesmo assim, obteve boas médias nas métricas observadas.

Ao se considerar as médias de latência não comparamos qual a transação o banco de dados está fazendo. Avaliando o tipo de transação, podemos detectar se um dos candidatos tem melhor desempenho em *inserts* ou *selects*, por exemplo. O YCSB fornece como saída informações de latência que discriminam o tipo de transação realizado, permitindo essa análise.O Gráfico 6 mostra a média de latência, agora considerando o tipo de transação que o banco de dados está executando.

Gráfico 6 – Latência média do teste, discriminada por tipo de transação, obtida no *benchmark* YCSB ao decorrer do teste para os produtos



O autor (2018)

Analisando o Gráfico 6, pode-se inferir que não há diferença significante entre o tipo de execução executada. As transações utilizadas para medir estes parametros são muito pequenas, assim quando o BD alvo recebe é capaz de executa-la imediatamente. Desta forma, os tempos apresentados ficam limitados a latência de rede, o que resulta nas latências observadas estarem tão próximas. A diferença fica na ordem de microssegundos.

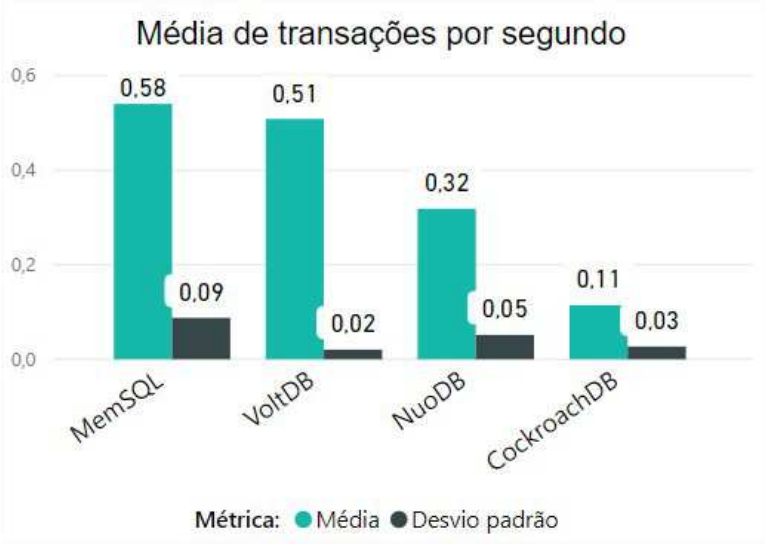
Apenas o CockroachDB oscila um pouco nas latências, se mostrando melhor executando a carga de deleção de registros e pior na inserção. Os demais mantêm uma média de 0,50 segundo de latência para todos os tipos de transação.

5.2 Resultados para o *benchmark* Voter

Na avaliação do *benchmark* Voter, utilizamos os parâmetros mencionados na Seção 4.1.3.1. Observamos os resultados em um teste limitado a 300 segundos, utilizando um fator de escala 10. O resultado foi coletado com dados de 5 execuções deste teste para cada produto.

No gráfico abaixo, analisamos a primeira métrica, do número de transações executadas por segundo ao decorrer do teste.

Gráfico 7 – Média de transações por segundo obtida para cada produto no *benchmark* Voter

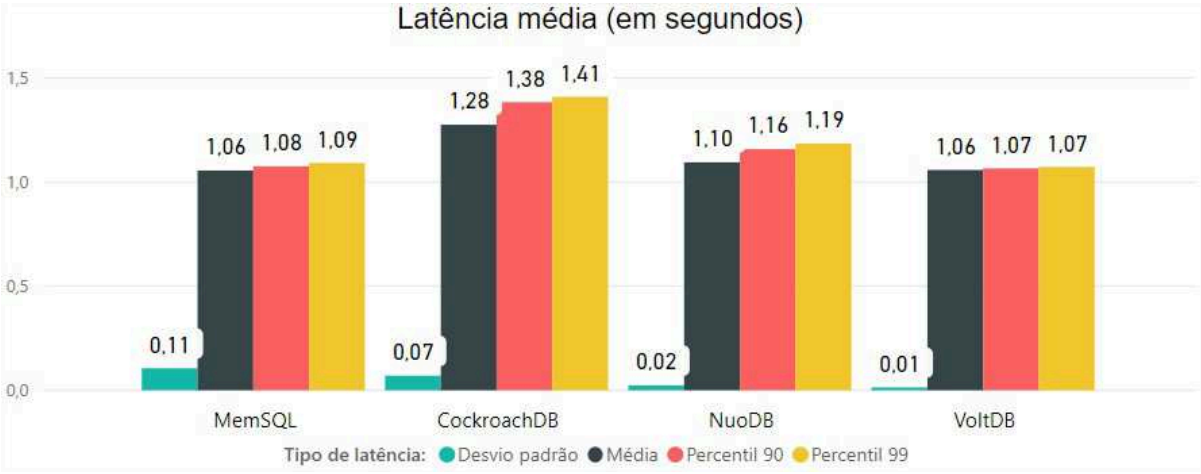


O autor(2018)

Analisando o Gráfico 7, novamente temos o MemSQL com melhores resultados na categoria, executando em média 0,58 transações a cada segundo do teste, seguido do VoltDB, que executou 0,51. NuoDB ficou intermediário entre o segundo e o último lugar, novamente ocupado pelo CockroachDB. O desvio padrão mostra que o grau de dispersão entre os resultados foi muito baixo, mostrando uma boa uniformidade na execução.

Logo após, analisamos a latência média obtida pelos produtos. O gráfico abaixo mostra os resultados obtidos pelos concorrentes:

Figura 9 – Latência média total nas transações obtidas para cada produto no *benchmark* Voter



O autor(2018)

Observamos novamente uma maior latência obtida pelo produto CockroachDB. O desvio padrão mostra que o grau de dispersão entre os resultados foi muito baixo,

mostrando uma boa uniformidade na execução. No teste do Voter temos uma situação muito diferente da observada no teste de latência média geral com YCSB no Gráfico 5. No Voter a variação obtida entre um produto e outro é pequena, tanto nas análises médias quanto nas de percentil 90 e 99, o que sugere uma distribuição uniforme entre as latências no teste. Isso demonstra que em transações menores, como é o caso deste *benchmark*, os produtos operam de forma semelhante.

O Voter visa uma exploração mais acentuada das características que se espera de um banco NewSQL. Como apresentado anteriormente, suas transações consistem em uma votação por número de telefone em um participante do "American Idol". Assim, este *benchmark* acarreta um estresse maior sobre os produtos testados, simulando diversos usuários votando em seu participante favorito. Os BDs MemSQL e VoltDB apresentaram resultados muito semelhantes, demostrando que são capazes de lidar com várias requisições simultâneas com baixa latência. O NuoDB apresenta uma arquitetura com níveis mais complexos de armazenamento, o que acaba prejudicando um pouco seu desempenho com um grande volume de transações mais simples. Já o Cockroach apresentou o pior resultado. Sua arquitetura, apesar de trazer novos algoritmos ainda fica limitada a algumas operações que utilizam o disco, o que deprecia muito seu desempenho em comparação aos demais.

5.3 Resultados para o *benchmark* TPC-H

O TPCH é um benchmark consolidado na avaliação de BDs com arquitetura distribuída na nuvem. Ele é o mais robusto dos três benchmarks analisados e possui em sua composição consultas OLTP e OLAP. Seu maior foco é em consultas mais complexas envolvendo contagens e somas para relatórios gerenciais (processamento OLAP). Apesar de sistemas NewSQL terem maior foco em OLTP, neste trabalhamos consideramos este *benchmark* para avaliar se as soluções propostas são flexíveis o suficiente para se adaptarem a cargas OLAP. O resultado foi coletado com dados de 3 execuções deste teste para cada produto.

Apenas foi possível obter os resultados de execução para o BD MemSQL. Os resultados apresentados nos gráficos a seguir demonstram que este BD não é totalmente preparado para este tipo de cargas. O Gráfico 10 demonstra que o BD foi capaz de tratar 1,4 transações por segundo. O desvio padrão mostra que o grau de dispersão entre os resultados foi médio. Porém o Gráfico 10 demonstra que a latência foi muito superior do que a observada nos experimentos com os demais *benchmarks*. Isso se deve ao fato das consultas apresentarem um nível de complexidade maior, apresentado somas e contabilizações sobre os dados.

No gráfico abaixo analisamos a primeira métrica, do número de transações

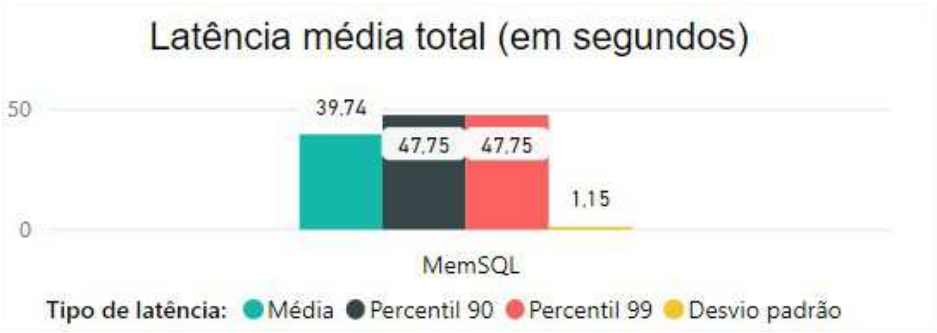
executadas por segundo ao decorrer do teste.

Figura 10 – Média de transações por segundo obtida no *benchmark* TPC-H ao decorrer do teste para o produto MemSQL

O autor(2018)

O resultado de análise da latência média é mostrado no gráfico abaixo.

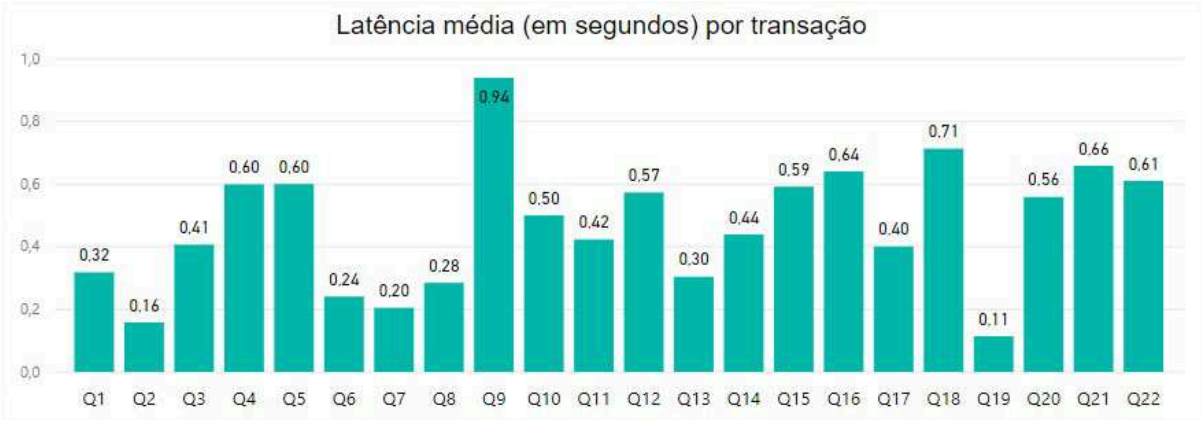
Gráfico 8 – Latência média total nas transações obtidas para o produto MemSQL no *benchmark* TPC-H



O autor (2018)

O Gráfico 9 apresenta a latência para cada uma das 22 consultas aplicadas pelo *benchmark*. O desvio padrão mostra que o grau de dispersão entre os resultados foi médio. Consultas como Q2, Q6, Q7,Q8 e Q19 apresentam seleções mais simples e mais restritivas, com baixa complexidade, permitindo uma latência menor. Já consultas como Q9 e Q18 são consultas mais complexas e apresentam filtros menos restritivos aplicando as suas operações em um conjunto maior de dados.

Gráfico 9 – Latência média do teste, discriminada por transação executada, obtida no *benchmark* TPC-H ao decorrer do teste para o MemSQL



O autor(2018)

Como mencionado anteriormente, foi possível executar este *benchmark* apenas para o MemSQL o que demonstra uma alta flexibilidade neste sistema. Nas tentativas de execução realizadas para o VoltDB esbarrou-se em uma limitação técnica imposta pelo próprio fornecedor. Como sua arquitetura é otimizada para execução de transações OLTP, o sistema não permite que uma tabela resultado ultrapase 1 GB (100 MB é o valor padrão), o que acarreta em erro e aborto da transação. O NuoDB realizou a importação dos dados rapidamente, porém travava na execução das consultas sem apresentar nenhum tipo de erro nos *logs* principais. O Cockroach não conseguiu realizar a carga dos dados, mesmo possuindo memória RAM suficiente para importar os 6GB do banco de dados do TPC-H, apresentou consumo muito elevado de memória esgotando os 8GB disponíveis no nodo para fazer esse carregamento, travando o nodo.

6 Conclusões

Nos últimos anos, a demanda cada vez maior por informação e o desafio de entregar essa informação de forma cada vez mais rápida, em sistemas geralmente *online* e com execução ininterrupta, motivou o desenvolvimento de tecnologias na área de bancos de dados e diversas outras, de forma a suprir tais necessidades. O próprio paradigma NewSQL, objeto de análise deste trabalho, foi criado com a finalidade de suprir as necessidades modernas, mantendo propriedades ACID e compatibilidade com sistemas existentes mantendo como linguagem de consulta a SQL. Isso permite que sistemas possam entregar informação rapidamente, processando cargas OLTP e OLAP em uma estrutura moderna e facilmente escalável, compatível com as necessidades modernas.

O presente trabalho analisou quatro produtos que se baseiam no paradigma NewSQL. Abordou-se o contexto acerca do paradigma NewSQL e foram explanadas características pertinentes de cada um dos produtos. Realizou-se a instalação de um ambiente distribuído em cluster físico com 3 nodos, de forma de comparar o comportamento das transações, nas nuances de cada uma das arquiteturas.

Para esta análise, empregou-se a técnica de benchmark, utilizando de benchmarks de domínio específico, através de um *framework* chamado OLTP-Bench. Estes benchmarks envolvem um contexto de aplicação, o que demonstra de forma mais próxima da realidade o comportamento em um ambiente real. Foram escolhidos três benchmarks que apresentam cargas de diferentes tipos e complexidades, de modo a cobrir a análise do processamento das cargas por parte dos produtos.

Sobre os resultados obtidos das análises verificou-se que geralmente o produto MemSQL se manteve a frente nas características observadas, obtendo alta taxa de *throughput,* e baixa latência nos contextos analisados. Também foi o único produto que conseguiu executar as cargas do teste TPCH mostrando-se mais flexível que os demais produtos.

Os produtos VoltDB e NuoDB se comportaram de maneira semelhante na maioria dos resultados analisados, mesmo com as considerações sobre as restrições da versão grátis utilizada pelo NuoDB e uma possível influência da linguagem de construção do VoltDB. Mostraram no geral uma boa execução das cargas analisadas, porém não conseguiram rodar o TPC-H para fornecer resultados sobre suas capacidades OLAP.

O banco de dados CockroachDB mostrou os piores resultados, com discrepâncias muito consideráveis nos contextos observados, principalmente nas taxas médias de transações por segundo. Os resultados com o CockroachDB se mostram justificáveis

quando observado o fato que o componente interno RocksDB e o *cache* interno do CockroachDB alternam o uso da memória RAM e o disco para armazenamento e execução das transações, diferentemente dos outros produtos que utilizam a memória RAM como memória principal para todas as operações. O banco de dados também não completou a execução dos testes TPC-H, afetando a análise OLAP do produto.

Os bancos de dados analisados mostram também que produtos NewSQL são uma alternativa interessante aos bancos de dados NoSQL, conseguindo entregar boas métricas de performance, mantendo as características ACID e resolvendo assim o impasse do teorema CAP de forma satisfatória. A redução natural no preço das tecnologias, como memória RAM, com o tempo tornarão esses produtos ainda mais atrativos, incentivando seu uso a problemas que utilizavam do paradigma NoSQL como alternativa.

Em vista dos resultados e conclusões obtidos com este trabalho, mesmo que ao encontro dos objetivos propostos, há possibilidades de complementá-lo, levantando então as seguintes sugestões de trabalhos futuros:

- Utilização de um benchmark adicional que possa avaliar o comportamento dos produtos para as capacidades no processamento OLAP dos bancos de dados observados;
- Realizar uma análise semelhante utilizando particionamento geográfico dos nodos para uma verificação mais aprofundada das características analisadas;
- Complementar o resultado com a criação de métricas auxiliares e monitoramento de outras dimensões, como, por exemplo, a utilização dos recursos de hardware pelo sistema operacional, e a criação de *threads* na execução, para agregar o entendimento sobre os resultados obtidos.
- Considerar a avaliação de bancos de dados de outros paradigmas envolvendo as mesmas métricas, para afirmar e comprovar as vantagens que o paradigma NewSQL propõe frente aos paradigmas relacional e/ou NoSQL.

Referências

ABADI, D. J. Data Management in the Cloud: Limitations and Opportunities. *IEEE Data Eng. Bull.*, v. 32, n. 1, p. 3 – 12, 2009. Disponível em: <http://sites.computer.org/debull/A09mar/abadi.pdf>.

BARATA, M.; BERNARDINO, J.; FURTADO, P. YCSB and TPC-H: Big data and decision support benchmarks. *2014 IEEE International Congress on Big Data*, IEEE, Anchorage, AK, USA, p. 800 – 801, Julho 2014. ISSN 2379-7703. Disponível em: <https://ieeexplore.ieee.org/document/6906872>. Acesso em: 20/08/2018.

BERNSTEIN, P. et al. The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (The Fully Redundant Case). *IEEE Transactions on Software Engineering*, IEEE, SE-4, n. 3, p. 154 – 168, Maio 1978. ISSN 1939-3520. Disponível em: <https://ieeexplore.ieee.org/document/1702517>. Acesso em: 13/10/2018.

CATTELL, R. Scalable SQL and NoSQL data stores. *SIGMOD Record*, v. 39, n. 4, p. 12 – 27, 2011. Disponível em: <http://portal.acm.org/citation.cfm?doid=1978915.1978919>.

CHEVALIER, M. et al. Benchmark for OLAP on NoSQL technologies comparing NoSQL multidimensional data warehousing solutions. In: *2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS) IS* -. [S.l.: s.n.], 2015. p. 480 – 485.

COELHO, F. et al. HTAPBench: Hybrid Transactional and Analytical Processing Benchmark. p. 293 – 304, 2017. ISSN 978-1-4503-4404-3.

COOPER, B. F. et al. Benchmarking Cloud Serving Systems with YCSB. *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*, Santa Clara, CA, USA, v. 1, n. 1, p. 143 – 154, Junho 2010. ISSN 9781450300360. Disponível em: <http://portal.acm.org/citation.cfm?doid=1807128.1807152>. Acesso em: 18/07/2017.

CURINO, C. A. et al. Benchmarking OLTP/web databases in the cloud: the OLTP-bench framework. *CloudDB '12 Proceedings of the fourth international workshop on Cloud data management*, ACM, Nova Iorque, v. 1, p. 17 – 20, Outubro 2012. ISSN 978-1-4503-1708-5. Disponível em: <https://dl.acm.org/citation.cfm?id=2390025>. Acesso em: 20/10/2018.

DEWITT, D. J. et al. Implementation techniques for main memory database systems. *SIGMOD '84 Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, ACM, Boston, Massachusetts, v. 14, p. 1 – 8, Junho 1984. ISSN 10.1145/971697.602261. Disponível em: <https://dl.acm.org/citation.cfm?id=602261>. Acesso em: 12/10/2018.

DIFALLAH, D. E. et al. OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *Proceedings ofthe VLDB Endowment*, v. 7, n. 4, p. 277 – 288, 2013. ISSN 21508097. Disponível em: <http://www.vldb.org/pvldb/vol7/p277-difallah.pdf>. Acesso em: 20/08/2017.

ELIAS, D. *O que significa OLTP e OLAP na prática?* 2016. Disponível em: <https://canaltech.com.br/business-intelligence/o-que-significa-oltp-e-olap-na-pratica/>. Acesso em: 06/10/2018.

FATIMA, H.; WASNIK, P. K. Comparison of SQL, NoSQL and NewSQL Databases for Internet of Things. *2016 IEEE Bombay Section Symposium (IBSS)*, p. 1 – 6, 2016.

GARCIA-MOLINA, H.; LIPTON, R. J.; VALDES, J. A Massive Memory Machine. *IEEE Transactions on Computers*, IEEE, C-33, n. 5, p. 391 – 399, Maio 1984. ISSN 0018-9340. Disponível em: <https://ieeexplore.ieee.org/document/1676454>. Acesso em: 12/10/2018.

GRAY, J. *Concurrency Control and Recovery in Database Systems*. Berlin: Springer, 1978. ISSN 0302-9743. ISBN 978-3-540-08755-7. Disponível em: <https://doi.org/10.1007/3-540-08755-9>. Acesso em: 20/10/2018.

GRAY, J. Database and Transaction Processing Performance Handbook. In: *The Benchmark Handbook*. [S.l.: s.n.], 1993.

GROLINGER, K. et al. Data management in cloud environments: NoSQL and NewSQL data stores. *Journal of Cloud Computing: Advances, Systems and Applications*, v. 2, n. 1, 2013. ISSN 2192-113X.

GUREVICH, Y. Comparative Survey of NoSQL/ NewSQL DB Systems. n. December, 2015.

JELLY, I.; KERRIDGE, J.; BATES, C. Benchmarking parallel SQL database machines. v. 826, p. 105 – 120, 1994. Disponível em: <http://liinwww.ira.uka.de/csbib?query=%2Bau:Jelly*%2Bau:Jelly&maxnum=200&sort=year>.

KARAMBIR, K.; MONIKA, S. Performance evaluation of NewSQL databases - IEEE Conference Publication. p. 1 – 5, 2017. Disponível em: <http://ieeexplore.ieee.org/document/8068585/#full-text-section>.

KAUR, K.; SACHDEVA, M. Performance Evaluation of NewSQL Databases. *International Conference on Inventive Systems and Control*, p. 1 – 5, 2017.

LABS, C. *Architecture Overview*. 2018. Disponível em: <https://www.cockroachlabs.com/docs/stable/architecture/overview.html#goals-of-cockroachdb>. Acesso em: 21/05/2018.

LANEY, D. 3D data management: Controlling data volume, velocity and variety. *META Group Research Note 6.70*, p. 1 – 4, 2001.

MAJDALANY, M. *What Is TPC*. 2001. Disponível em: <http://www.tpc.org/information/about/abouttpc.asp>. Acesso em: 29/04/2018.

MEMSQL. *MemSQL Architecture: Technology Innovations Power Convergence of Transactions and Analytics*. 2018. Disponível em: <https://www.memsql.com/content/architecture/>. Acesso em: 06/10/2018.

MOHAN, C. et al. ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Transactions on Database Systems (TODS)*, ACM, Nova Iorque, v. 17, n. 1, p. 94 – 162, Março 1992. Disponível em: <https://dl.acm.org/citation.cfm?id=128770>. Acesso em: 20/10/2018.

NUODB INC. *NUODB: ENTERPRISE SQL AND ARCHITECTED FOR THE CLOUD*. 2018. Disponível em: <https://www.nuodb.com/product-overview>. Acesso em: 12/05/2018.

NUODB INC. *NuoDB Pricing & Editions*. 2018. Disponível em: <https://www.nuodb.com/product/pricing-editions>. Acesso em: 03/05/2018.

PATGIRI, R.; AHMED, A. Big Data: The V's of the Game Changer Paradigm. In: IEEE (Ed.). *2016 IEEE 18th International Conference on High Performance Computing and Communications*. Sydney: [s.n.], 2016. p. 17 – 24. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7828355&isnumber=7828341>. Acesso em: 06/05/2018.

PAVLO, A. What's Really New with NewSQL? *SIGMOD Record*, v. 45, n. 2, 2016. Disponível em: <http://dl.acm.org/citation.cfm?doid=3003665.3003674>.

RUDOLL, C. SQL, noSQL or newSQL – comparison and applicability for Smart Spaces. n. May, p. 31 – 37, 2017.

SADALAGE, P. J.; FOWLER, M. *NoSQL Distilled - A brief guide to the emerging world of polyglot persistence*. 1. ed. [S.l.]: Pearson Education, 2013.

SHAMSUDDIN, S. M.; HASAN, S. Data science vs big data @ UTM big data centre. p. 1 – 4, 2016.

SHI, X. et al. Research on Supply Chain Information Classification Based on Information Value and Information Sensitivity. In: *2007 International Conference on Service Systems and Service Management*. Chengdu, China: IEEE, 2007. p. 1 – 7. ISBN 1-4244-0884-9. ISSN 2161-1890. Disponível em: <https://ieeexplore.ieee.org/document/4280248>. Acesso em: 01/11/2018.

SHIM, S. S. The CAP Theorem's Growing Impact. *Computer*, IEEE, San Jose, v. 2, p. 21 – 22, 45 2012. ISSN 1558-0814. Disponível em: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6155651>. Acesso em: 20/09/2018.

STANCU-MARA, S.; BAUMANN, P. A Comparative Benchmark of Large Objects in Relational Databases. In: *Proceedings of the 2008 International Symposium on Database Engineering & Applications*. New York, NY, USA: ACM, 2008. (IDEAS '08), p. 277 – 284. ISBN 978-1-60558-188-0. Disponível em: <http://doi.acm.org/10.1145/1451940.1451980>.

STEPPAT, N. *NoSQL – Do teorema CAP para P?(A|C):(C|L)*. 2011. Disponível em: <http://blog.caelum.com.br/nosql-do-teorema-cap-para-paccl/>. Acesso em: 20/08/2018.

STONEBRAKER, M. *New SQL: An Alternative to NoSQL and Old SQL for New OLTP Apps*. 2011. Disponível em: <https://cacm.acm.org/blogs/blog-cacm/109710-new-sql-an-alternative-to-nosql-and-old-sql-for-new-oltp-apps/fulltext>. Acesso em: 15/11/2017.

STONEBRAKER, M. *New SQL: An Alternative to NoSQL and Old SQL for New OLTP Apps*. 2011. Disponível em: <https://cacm.acm.org/blogs/blog-cacm/109710-new-sql-an-alternative-to-nosql-and-old-sql-for-new-oltp-apps/fulltext>. Acesso em: 15/11/2017.

TRANSACTION PROCESSING PERFORMANCE COUNCIL (TPC). *TPC BenchmarkTM H Standard Specification Revision 2.17.3*. San Francisco, CA, USA, 2017. Documentação técnica. Disponível em: <http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.3.pdf>. Acesso em: 18/05/2018.

USC DATABASE LAB. *Overview - BG Benchmark*. 2015. Disponível em: <http://www.bgbenchmark.org/BG/overview.html>. Acesso em: 20/10/2018.

VOLTDB. *VoltDB Technical Overview*. 2013. Disponível em: <http://www.odbms.org/wp-content/uploads/2013/11/VoltDBTechnicalOverview.pdf>. Acesso em: 20/10/2018.

VOLTDB, INC. *Using VoltDB*. 2018. Disponível em: <http://downloads.voltdb.com/documentation/UsingVoltDB.pdf>. Acesso em: 06/05/2018.

Apêndices

APÊNDICE A – Instalação e configuração do produto VoltDB

Os passos a seguir orientam a instalação do produto VoltDB no ambiente deste trabalho. É necessário atentar aos comandos, que estão em sintaxe de funcionamento no sistema operacional Xubuntu, como previamente mencionado na seção 4.2.

Modo de instalação: Docker (para mais informações, consultar seção 4.1.5)

Versão corrente do produto utilizada: VoltDB Community Edition 8.2

1) Deve ser utilizado o comando “docker pull voltdb/voltdb-community” no terminal para orientar o sistema a realizar o *download* da versão corrente do VoltDB Community Edition no Docker Hub[1];
2) Defina qual será o nodo mestre;
3) Considerando que os nós estão em máquinas fisicamente separadas, os mesmos podem ser conectados via rede docker para que seja possível encontrá-los na rede. Para isso foram utilizados por este autor dois recursos do Docker: O Swarm[2], e uma rede operando no modo Overlay[3].
    1) O Swarm já acompanha o Docker como padrão. Basta executar o comando, no nodo mestre: “docker swarm init” no terminal. Este comando irá iniciar este nodo como administrador e irá gerar um comando, no terminal, que deve ser copiado e reproduzido nos outros nodos para que eles se unam a rede;
    2) Após Swarm configurado, executar o comando no nodo mestre: “docker network create -d overlay –attachable <nome_da_rede>“ para criar uma rede no modo overlay. Note, a variável <nome_da_rede> deve ser substituída pelo nome da rede que se deseja criar;
    3) Para iniciar um host, comece iniciando o host mestre com o comando: “docker run -d -P -e HOST_COUNT=3 -e HOSTS=node1,node2,node3 –name=node1 –network=<nome_da_rede> <volt_img>“.
        1) A variável HOST_COUNT foi configurada neste trabalho para um ambiente de 3 hosts, mas pode ser configurado par amais e para menos. Ao mudar essa quantidade, deve-se ajustar a variável HOST com o nome dos nodos.
        2) A variável <nome_da_rede> deve ser preenchida com o nome da rede que você criou;

[1] https://hub.docker.com/r/voltdb/voltdb-community/
[2] https://docs.docker.com/engine/swarm/
[3] https://docs.docker.com/network/overlay/

3) Para este trabalho, a versão corrente do produto VoltDB era 8.2. Depois de fazer o download da imagem docker, você pode inserir o comando “docker image ls” e ver o nome da versão que foi baixada, e preencher a variável <volt_img> com este valor.

4) Executar o mesmo comando de incialização para os outros nodos, mudando a variável –name para o nome do nodo que se está executando o comando.

4) O arquivo de inicialização do docker foi mudado antes da execução para suportar as tabelas temporárias criadas pelo *benchmark* TPC-H. as variáveis foram mudadas são: <temptables maxsize=“2500”/> e <query timeout=“2147483647”/>.

5) A interface Web pode ser acessada via endereço: http://localhost:8080.

APÊNDICE B – Instalação e configuração do prouto CockroachDB

Os passos a seguir orientam a instalação do produto CockroachDB no ambiente deste trabalho. É necessário atentar aos comandos, que estão em sintaxe de funcionamento no sistema operacional Xubuntu, como previamente mencionado na seção 4.2.

Modo de instalação: Tradicional/local

Versão corrente do produto utilizada: CockroachDB v2.1.0

1) Fazer o download dos binários, seguindo a orientação do fabricante na página de download oficial[1];

2) Após executar a instalação via instruções da página oficial citada acima, o comando abaixo inicia um cluster no modo inseguro, com parâmetros de memória para transações e cache aumentado para dar suporte aos testes.

3) Executar comando abaixo de uma vez só para iniciar um membro do cluster. Note que a variável <this_host_ip> deve ser substituída pelo endereço do host em que se executa o comando e as variáveis <other_host_ip_X> devem ser preenchidas com os endereços IP dos outros membros do cluster, tanto quanto existirem.

    1) cd <cockroach_path>, onde <cockroach_path> é o caminho da pasta do CockroachDB
    2) ./cockroach start \
       –insecure \
       –host=<this_host_ip> \
       –join=<other_host_ip_1>, <other_host_ip_2>, <....>\
       –max-sql-memory=1024MiB \
       –cache=1024MiB

4) Executar o comando nos outros nós do cluster, adaptando as variáveis;

5) Após iniciada, a interface Web pode ser acessada via endereço: http://localhost:8888

[1] https://www.cockroachlabs.com/docs/stable/install-cockroachdb.html

6) Para criar a base de dados para o teste, pode se utilizar o comando a seguir em qualquer um dos nodos: “./cockroach sql –host= <any_cluster_host_ip> –insecure -e ’CREATE DATABASE <nome _base> ’“ , substituindo a variável <any _cluster_host_ip> por um endereço IP de um dos membros do cluster, e <nome_base> pelo nome da base de dados que se deseja criar.

APÊNDICE C – Instalação e configuração do produto MemSQL

Os passos a seguir orientam a instalação do produto MemSQL no ambiente deste trabalho. É necessário atentar aos comandos, que estão em sintaxe de funcionamento no sistema operacional Xubuntu, como previamente mencionado na seção 4.2.

Modo de instalação: Tradicional/local

Versão corrente do produto utilizada: MemSQL Developer Edition 6.5

1) O modo de instalação do MemSQL difere um pouco dos outros produtos. Considerando que os nodos estão em uma mesma rede local lógica, é instalado o produto em um nodo (mestre), e este fará a instalação automaticamente nos outros nodos.

    1) Para download do produto, seguir passos da documentação oficial do produto[1];
    2) Acessar a interface na máquina mestre, via navegador Web, no endereço http://localhost:9000
    3) No primeiro acesso a interface irá questionar se você deseja utilizar o produto no modo Single Host ou não. Caso queira configurar o cluster, selecione a opção não;
    4) Será questionado sobre os endereços dos computadores na rede a ser usados como nodos folha. Preencha os campos e aguarde o teste de conexão;
    5) Se a conexão exigir uma senha, digite. Caso contrário, a instalação será feita automaticamente pelo MemSQL, e ao terminar os nodos estarão disponíveis no *dashboard* da página principal.

[1] https://docs.memsql.com/installation/v6.5/full-installation-guide/

APÊNDICE D – Instalação e configuração do produto NuoDB

Os passos a seguir orientam a instalação do produto NuoDB no ambiente deste trabalho. É necessário atentar aos comandos, que estão em sintaxe de funcionamento no sistema operacional Xubuntu, como previamente mencionado na seção 4.2.

Modo de instalação: Tradicional/local

Versão corrente do produto utilizada: NuoDB Community Edition 3.3.0.1

1) Executar o download da versão corrente na página de download do NuoDB[1];

2) O arquivo estará na forma de executável e deve ser instalado via dkpg ou no modo gráfico[2];

3) Após instalação, deve-se modificar a configuração de host. Primeiro é feito no nó mestre, que deve ser eleito por você.

   - sudo vi /opt/nuodb/etc/default.proprieties
   - alterar a variável domainPassword para uma senha de sua escolha. Ela será a senha de domínio.
   - Alterar a variável peer para “localhost”
     - Nos outros nodos, a mesma configuração deve ser feita, e nesta variável peer, deve ser informado o endereço IP do host mestre;

4) Para iniciar o produto, use o comando: “sudo /opt/etc/nuoagent start

5) Executar passos 1,2,3 e 4 em todos os hosts, observando as informações do passo 3;

6) Execute o comando no nó mestre: “/opt/nuodb/bin/nuodbmgr –broker localhost –password <a senha setada no arquivo do passo 3>

   - Este comando dá acesso ao controle de domínio, onde são gerenciadas as bases do NuoDB.

7) Após entrar no modo de domínio, execute o comando “create database” para criar uma base gerenciada, e informe apenas as informações dos parâmetros abaixo, podendo avançar as outras opções apertando a tecla ENTER.

[1] https://www.nuodb.com/dev-center/community-edition-download

[2] http://doc.nuodb.com/Latest/Content/Using-the-Debian-Package-Utility-to-Install-NuoDB.htm?tocpath=Deployment%20Models%7CDeploying%20NuoDB%C2%A0in%20Physical%20or%20VMware%20Environments%7CInstalling%20NuoDB%7CLinux%20Installation%7C_____2

1) Database name: Coloque o nome da base;
2) Template : digite “Multi Host”;
3) Escape as outras opções com a tecla ENTER;

8) Pronto, o NuoDB está instado em ambiente distribuído, com uma base para conexão

APÊNDICE E - Artigo

Análise e benchmarking das soluções NewSQL CockroachDB, MemSQL, NuoDB e VoltDB

Ronan Romeu Knob [1]

[1] Depto. de Informática e Estatística – Universidade Federal de Santa Catarina - UFSC
Campus Reitor João David Ferreira Lima, s/n - Trindade, Florianópolis - SC, 88040-900

ronanknob@grad.ufsc.br

Abstract. *This paper presents an experiment that compares 4 databases based on the NewSQL paradigm. The method used was benchmarking, employing 3 specific domain benchmarks and a framework for execution. Performance characteristics of the systems and their adequacy to the characteristics of the NewSQL paradigm are evaluated. This analysis can contribute as a reference for future uses of the technology.*

Resumo. *Este artigo apresenta um experimento que compara 4 bancos de dados baseados no paradigma NewSQL. O método utilizado foi o* benchmarking, *empregando 3* benchmarks *de domínio específico e um* framework *para execução. São avaliadas características de desempenho dos sistemas e a sua adequação ás características do paradigma NewSQL. Esta análise pode contribuir como referência para futuros usos da tecnologia.*

1. Introdução

Os avanços em tecnologias Web e na proliferação de dispositivos móveis e conectados á internet, criaram no âmbito da informática uma necessidade de armazenamento de grandes quantidades de dados. O primeiro paradigma a tentar organizar esses dados foi o relacional. O aumento exponencial nos volumes de dados armazenados com o passar dos anos, e, ao mesmo tempo, a necessidade de recuperar informações com agilidade, fizeram emergir novos paradigmas em bancos de dados como é o caso do NoSQL (Not Only SQL). Tais paradigmas foram importantes, mas sacrificavam características dos bancos tradicionais, como a normalização e integridade. Os problemas com os paradigmas anteriores levaram ao surgimento do paradigma NewSQL.

O paradigma NewSQL visa usufruir dos benefícios do paradigma relacional com a escalabilidade do paradigma NoSQL. São SGBDs modernos, que buscam prover o mesmo desempenho escalável do NoSQL, para cargas de trabalho OLTP (Online Transactional Processing) de leitura e escrita, enquanto mantém as

garantias ACID para as transações [Pavlo, 2016]. Neste trabalho, comparamos 4 produtos baseados neste paradigma, através de 3 tipos de benchmarks, de forma a avaliar características pertinentes desses produtos e do paradigma NewSQL.

2. Trabalhos relacionados

Este capítulo apresenta um resumo de três trabalhos que foram usados como fundamentação para este trabalho. Os mesmos buscam explanar os conceitos ligados ao paradigma NewSQL e aplicações dos produtos.

Em um dos primeiros trabalhos que aborda o paradigma NewSQL, Pavlo [Pavlo, 2016] faz uma análise do histórico dos sistemas de banco de dados, e a crescente necessidade de armazenamento da informação é realizada. Há informações chaves como a introdução da ideia de que a aplicação e os dados deveriam rodar em separado, e as dificuldades na adaptação das empresas em servir dados na velocidade que as aplicações Web que foram surgindo necessitaram. Além disso, são apresentadas as características esperadas desses produtos.

Da análise geral do trabalho pode-se dizer que o paradigma NewSQL não apresenta uma mudança radical em nenhum conceito abordado. O paradigma se apresenta mais como um apanhado das melhores características dos paradigmas anteriores. As características chave não possuem conceitos novos, mas os produtos baseados neste paradigma trazem uma implementação que traz todos esses conceitos juntos, em produtos desenhados para trabalhar numa arquitetura distribuída. Em adição, mantém a compatibilidade com a linguagem de consulta SQL, o que permite que seja feita a migração de sistemas legados para trabalhar com desempenho e integridade numa arquitetura moderna. De mesma forma se mostra interessante aos sistemas novos, sendo sistemas que permitem escalabilidade horizontal e elasticidade dinâmica.

O artigo de Karambir Kaur e Monika Sachdeva [Kaur e Sachdeva, 2017] apresenta uma avaliação de desempenho entre bancos do paradigma NewSQL. Os autores avaliaram as mesmas 4 escolhas de produtos elegidos para este trabalho. O objetivo do trabalho consistiu em estudar as soluções selecionadas e questionar se existiria suficiente conhecimento das qualidades dos sistemas NewSQL para ajudar um engenheiro de software no processo de decisão. O método de teste escolhido pelos autores analisou as operações de create, insert, update e select nos bancos de dados selecionados, para medir as características de tempo de execução, latência de leitura, latência de escrita, e latência de atualização.

Por fim, na dissertação o autor Yuri Gurevich [Gurevich, 2015] faz um benchmarking entre soluções dos paradigmas NoSQL e NewSQL. A escolha dos produtos selecionados para o teste foi baseada numa análise de tendência feita pelo autor pela fonte DB-Engines[1], uma das fontes também utilizadas neste trabalho. Foi adotado também o método de *benchmarking*, usando para isso o *benchmark* BG[2]. Para o teste , Cassandra foi o produto NoSQL usado na compa-

[1] https://db-engines.com/en/

[2] http://www.bgbenchmark.org/BG/overview.html

ração, baseado em sua popularidade e por ser construído usando vários conceitos de dois notáveis projetos NoSQL: Dynamo[3] e BigTable[4]. Como representante do paradigma NewSQL, foi escolhido o NuoDB,por implementar internamente varias técnicas que foram citadas na fundamentação, como MVCC e replicação assíncrona.

Nas métricas avaliadas por Karambir Kaur e Monika Sachdeva [Kaur e Sachdeva, 2017] foram utilizadas métricas simples, que podem ajudar no processo de escolha de uma solução, mas não necessariamente são as melhores métricas para se trabalhar com esse tipo de banco, visto que muitos dos benefícios do paradigma NewSQL são observados em ambiente distribuído. Já na avaliação de Yuri Gurevich [Gurevich, 2015], temos o emprego de um benchmark na avaliação, e métricas mais robustas, como o índice SoAR, métrica oficial do benchmark utilizado. Entretanto, a comparação entre produtos de diferentes paradigmas tende a mascarar a avaliação das características de cada paradigma.

Por fim temos a configuração do ambiente de testes. No trabalho mostrado de Karambir Kaur e Monika Sachdeva [Kaur e Sachdeva, 2017] foi utilizada apenas uma instância física, podendo não avaliar com clareza as características de paralelismo dos sistemas selecionados. No texto de Yuri Gurevich [Gurevich, 2015] não há sequer informação sobre essa configuração no texto, o que torna difícil reproduzir o experimento. Selecionar a arquitetura pensando nos produtos a se testar é importante, pois como já mencionado, esses produtos têm vantagens relacionadas a ambientes de múltiplas instâncias, então uma comparação em um ambiente com um único nodo ou configuração insuficiente pode não representar um ambiente de uso real destes produtos.

3. Bancos de dados NewSQL

Este capítulo apresenta os candidatos selecionados para os testes experimentais. Em linhas gerais, os bancos classificados como NewSQL têm muitos dos requisitos para gerenciamento em ambientes de computação em nuvem (ou distribuída), que se popularizam cada vez mais, e também oferecem os benefícios dos padrões já conhecidos da SQL [Grolinger *et al.*, 2013]. Uma das principais vantagens de usar um SGBD que é construído para uma execução distribuída é que suas partes são construídas com foco em distribuição de informações, diferente do cenário onde os produtos são adaptados para tal funcionamento. Isso inclui possibilidades como otimizadores de consultas e protocolos de comunicação entre nodos [Pavlo, 2016].

Neste trabalho foram escolhidos exclusivamente produtos baseados no paradigma NewSQL, com versões de uso gratuito que permitissem os testes. As escolhas tiveram como critérios de seleção: O site ranking DB-Engines[5], especializado em bancos de dados; número de mencões em websites e interesse nas buscas (via Google Trends[6]); a análise da pesquisa anual do *website* Stack

[3]https://aws.amazon.com/pt/dynamodb/
[4]https://cloud.google.com/bigtable/
[5]db-engines.com/
[6]https://trends.google.com.br/trends/

Overflow no ano de 2017[7], e a adequação destes com requisitos esperados do paradigma. A pesquisa foi feita entre outubro e novembro de 2017. Também foi considerada a proposta dos sistemas em comparação com as características do paradigma NewSQL.

3.1. VoltDB

O VoltDB é um sistema de banco de dados desenvolvido por uma empresa que carrega o mesmo nome. Foi desenvolvido em 2014 pelos cientistas da computação: Dr. Michael Stonebraker, também criador do termo NewSQL, Sam Madden, e Daniel Abadi. Ela é disponibilizada em versões enterprise e community, sendo esta última sob licença GNU Affero General Public License [VoltDB, Inc, 2018].

O funcionamento geral do banco VoltDB utiliza as seguintes características:

- Uso de armazenamento em memória principal para maximizar a vazão de dados, previnindo custosos acessos em disco aos dados;
- Ganho de desempenho por serialização de acesso a todos os dados, previnindo o consumo de tempo das funções de locking, latching, logs de transação, etc, dos tradicionais SGBDs;
- Clusterização com replicação sob múltiplos servidores, que garantem escalabilidade, confiabilidade e alta disponibilidade dos dados.
- Compatível com as características ACID e uso da ANSI standard SQL, que garante uma rápida curva de aprendizado dos usuários avançados de bancos de dados e permite fazer as transações direto da aplicação.

Na arquitetura do VoltDB, os clusters contém uma fila de processamento, uma *engine* de execução e as tabelas com os dados indexados. A comunicação é feita entre nodos quando há necessidade de se fazer uma consulta que necessita dados de múltiplas partições. Neste cenário, um dos nodos age como coordenador e distribui o trabalho necessário entre os outros nodos, coletando os resultados e completando a tarefa [VoltDB, 2013].

3.2. NuoDB

O NuoDB lançou sua primeira versão em 2010. O produto é disponibilizado nas versões *Community*, que é gratuita, sob licença proprietária; Professional e Enterprise. A versão Community utilizada no trabalho inclui restrições de escalabilidade: Falando nos termos da arquitetura do NuoDB, a versão grátis permite 1 instância de administração, 1 SM(Storage Management) e 3 TE's (Transaction Processsing) para escalabilidade (na versão paga a escalabilidade pode ser r por ilimitados SM's e TE's). Os conceitos de SM e TE fazem parte da arquitetura e são explanados a seguir.

Transaction processing layer - A camada que recebe as requisições SQL. é uma camada que contém nodos em memória chamados *Transaction Engines* (TE's). Essas TE's permitem manter desempenho alto em memória.

[7]https://insights.stackoverflow.com/survey/2017

Quando uma aplicação faz requisições ao NuoDB, os TE's criam cache em memória para a carga de trabalho da aplicação. As requisições para arquivos que não estão em cache são alimentadas com caches de memória de outros TE's ou pela camada de gerenciamento de armazenamento.

The storage management layer - A segunda camada mostrada na imagem conectando com os dados físicos, consiste em nodos de processo chamados *Storage Managers* (SM's), que possuem tanto componentes em memória quanto guardados em disco. Os SM's também oferecem garantias de durabilidade dos dados. Múltiplos SM's podem ser usados para aumentar a redundância de dados.

As duas camadas acima descritas tornam o NuoDB escalável, pois necessita simplesmente adicionar ou remover TE's e SM's para dimensionar o banco conforme a necessidade. Com essa modularidade, há também a opção de escolher um modo de replicação para cada novo banco de dados criado.

3.3. CockroachDB

O CockroachDB foi lançado em 2015 por três ex-funcionários do Google: Spencer Kimball, Peter Mattis, and Ben Darnell. O projeto CockroachDB foi criado para ser um banco de dados open-source, distribuído no nível de que uma instância pode ser levantada em um computador pessoal comum e ajudar no processamento das requisições feitas [Labs, 2018].

O produto é distribuído nas versões Core e Enterprise, sendo a primeira grátis. O programa, tem código aberto para as duas versões. A versão Core é disponibilizada gratuitamente. Apesar de novo entre os bancos de dados do paradigma NewSQL, ele não utiliza armazenamento final em memória principal. Ao invés disso, é feito o aproveitamento de uma estrutura de clocks atômicos para escrita de blocos, que facilita a obtenção das características ACID nas transações. A arquitetura converte comandos SQL em dados Key-Value (KV), que são extremamente rápidos de manipular. Essa arquitetura é composta de 6 camadas que interagem entre si e suas partes são brevemente apresentadas a seguir.

- SQL Layer - A camada SQL recebe as consultas ao banco de dados via uma interface API, para qualquer nodo do cluster, uma vez que todos se comportam de maneira simétrica. A camada SQL repassa essa mensagem para a camada de estruturação, que será responsável por criar o plano de execução.
- Transaction Layer - A camada de transação é responsável por criar o plano de execução para as requisições SQL recebidas. A consulta e transformada num plano de execução. O plano é passado para a camada subsequente, um gerenciador de operações.
- Distribution Layer - A camada de distribuição é responsável por guardar um mapa classificado monolítico com os pares de chaves Key-Value, um espaço que descreve todos os dados do cluster e sua localização. O mapa é dividido em "intervalos", de modo que a consulta das chaves não fique centralizada em um nodo.

- Replication Layer - Essa camada é responsável por copiar os dados entre os nodos, e aplicar o algoritmo de “consensus” para garantir a consistência. Para garantir essa consistência, o algoritmo de “consensus” funciona de forma que necessite um quórum de nodos ativos para aceitar uma transação. O menor número possível para essa funcionalidade são três nodos ativos.
- Storage Layer - Cada nodo do CockroachDB possui ao menos um *store*, que é o espaço onde o processo do DB lê e escreve dados no disco. São guardados os dados através da API RocksDB[8], que guarda os valores key-value no disco.

Cada uma destas camadas que compõem a arquitetura do CockroachDB se comunica com a camada imediatamente inferior. A ligação nas camadas da arquitetura permite que o comando SQL chegue no armazenamento de chave-valor com sucesso.

3.4. MemSQL

O MemSQL teve sua primeira versão lançada em abril de 2013. É escrito na linguagem de programação C e distribuído nas versões *Developer*, que é grátis, mas não é recomendado o uso em produção e tem recursos limitados, e a versão *Enterprise*, paga, com todas as funcionalidades. Atua sob licença privada. Apresentado pelo fornecedor como um banco de dados relacional distribuído que suporta transações e análises em tempo real, em escala. É acessível pela sintaxe SQL, incluindo *joins*, filtros e capacidades analíticas (agregações, funções de janelamento, *group by*, etc).

A estrutura do MemSQL é basicamente composta de duas camadas: Os nodos agregadores (*Aggregator nodes*) e os nodos folha (*Leaf nodes*). Os nodos agregadores, funcionam como roteadores de consulta, e atuam como um *gateway* no sistema distribuído. Eles armazenam apenas metadados e dados de referência, e distribuem as consultas nos nós folha e agregam os resultados enviados de volta ao cliente. Já Os nodos folha (Leaf nodes) possuem a função de armazenar e computar as tarefas. Os dados são automaticamente distribuídos através dos nodos folha para as partições que fazem a execução paralelizada das consultas.

A comunicação entre os nodos é realizada via comandos SQL sob um protocolo MySQL[9]. As duas camadas da arquitetura possuem planos de recuperação automáticos em caso de falha para prover tolerância a falhas. A proporção de nodos agregadores e nodos folha na aplicação determina a capacidade e o desempenho do cluster, e pode variar conforme o domínio da aplicação ao qual se designam [MemSQL, 2018].

4. Análise experimental

Esta seção apresenta o *framework* OLTP-Bench, utilizado para executar os *benchmarks;* os critérios de seleção e os *benchmarks* que foram escolhidos, a arquitetura física do ambiente do teste e métricas avaliadas.

[8]https://rocksdb.org/

[9]https://dev.mysql.com/doc/internals/en/client-server-protocol.html

4.1. OLTP-Bench

O OLTP-Bench foi a ferramenta utilizada para avaliar os sistemas de bancos de dados neste trabalho. Ela é uma *testbed*, ou uma suíte de *benchmarking*. A ferramenta já traz suporte a vários sistemas de bancos de dados do mercado, assim como uma boa carga de *benchmarks* diferentes, e permite expansibilidade pelo usuário para englobar novos produtos e testes, caso deste trabalho.

O OLTP-Bench apresenta fácil uso, permite extensão das suas funcionalidades, faz um bom controle das transações, avaliação de requisições, e distribuição dos acessos [Difallah *et al.*, 2013]. A ferramenta mostra valor especialmente por três características: (i) Um *framework* automatizado e extensível de configurar, executar e analisar os resultados dos experimentos de desempenho do DBMS, com controladas e repetíveis configurações; (ii)Datasets reais e geradores sintéticos, com suas cargas de trabalho, todos implementados na mesma ferramenta; (iii) Descobertas experimentais sobre os DBMSs e DBaaS, resultante de centenas de experimentos.

Em sua arquitetura, temos o OLTP-Benchmark gerando uma fila de transações para execução de acordo com o *benchmark* e a emulação de usuários virtuais, chamados *Workers*, que atuam inserindo essas consultas geradas no pool JDBC dos bancos de dados. Durante a execução deste teste, o OLTP-Bench coleta as estatísticas de execução, e retorna um arquivo saída padrão com esses valores. Para *benchmarks* que tem estatísticas específicas é fornecido um arquivo adicional como saída. O OLTP-Bench ainda pode lidar dinamicamente mudando o número de *Workers* e pesos das transações, para uma modelagem mais complexa. Também pode rodar vários benchmarks ao mesmo tempo, para testar a habilidade do DBMS de balancear a carga sob diferentes circunstâncias.

4.2. Benchmarks

Um *benchmark* é um protocolo de testes que tem por objetivo produzir dados de desempenho que podem ser usados para fazer comparações significativas entre os sistemas testados [Jelly *et al.*, 1994].Considerando que os produtos NewSQL são concebidos para preencher a lacuna entre o desempenho do paradigma NoSQL e, ao mesmo tempo, garantir características ACID [Pavlo, 2016], duas premissas foram observadas na escolha do *benchmark* a ser utilizado: Dois candidatos deveriam simular um cenário com transações mais simples, porém em grande quantidade, e um último para avaliar as possibilidades de processamento de cargas OLAP, mesmo não sendo o foco do paradigma. Os *benchmarks* YCSB, Voter e TPC-H foram escolhidos nesses critérios, e sõa apresentados a seguir.

4.2.1. YCSB

O Yahoo! *Cloud Serving Benchmark* (YCSB) consiste em uma aplicação geradora de carga de trabalho e um pacote com cargas de trabalho padrão que cobrem interessantes partes da avaliação de desempenho (cargas de trabalho de leitura intensa, cargas de trabalho de escrita intensa, varredura de tabelas, etc.). Os conjuntos de cargas de trabalho usadas no YCSB recebem o nome de

YCSB *Core Package*. Cada carga de trabalho representa um misto de operações de leituras e escritas, usando diferentes volumes de dados e números de requisições.

A estrutura deste teste consiste em uma única tabela nomeada como usertable, com N campos. Cada registro é identificado como uma chave primária, algo como “user234123”. Cada campo é nomeado como field0, field1, e assim por diante. Os valores dos campos são *strings* randômicas de caracteres ASCII de tamanho aleatório. Os parâmetros de número de campos(N) e o fator de escala (F) são informados, por isso não constam como número fixo.

A execução do teste realiza muitas escolhas aleatórias gerando a carga do teste: As operações que serão feitas (dentre Insert, Update, Read or Scan), qual registro ler ou escrever, quantos registros examinar, e assim por diante. Essas decisões são governadas por distribuições randômicas, utilizando para isso as distribuições uniforme, distribuição de Zipf, distribuição baseada em registros recentes e distribuição Multinomial. Nos parâmetros definidos para á execução, configuramos fatores relacionados à escala de volume da base de teste e os dados da carga de trabalho. Utilizamos fator de escala 1000, 64 usuários virtuais emulados para manipulação, e um limite do teste de 300 segundos.

4.2.2. Voter

O Voter é um *benchmark* baseado em um software utilizado em um programa de talentos exibido em televisão, nos países do Japão e Canadá. Os usuários ligam para votar no seu candidato favorito. Ao receber a ligação a aplicação invoca a transação que atualiza o número total de votos de cada participante. O sistema de banco de dados então registra o número de votos feitos por cada usuário, fixado em um limite máximo. Uma transação em separado é periodicamente invocada para computar os votos totais durante o programa.

Este *benchmark* foi desenhado para saturar o banco de dados com pequenas transações, todas atualizando um pequeno número de registros. A arquitetura deste teste se dá através de três tabelas que guardam algumas informações sobre os candidatos e o usuário que está ligando, e duas *Views*, que são consultadas para atualizar o status no programa de televisão. Para execução, como parâmetros o fator de escala foi 10 e o número de usuários virtuais emulados foi 30.

4.2.3. TPC-H

O TPC-H avalia o desempenho de vários sistemas de apoio a decisão pela execução de um conjunto de consultas sob uma base em condições controladas. Em sua arquitetura, o esquema representa arquivos de um importador multinacional fictício e revendedor de partes e suprimentos industriais. Os clientes e fornecedores deste negócio vem de diferentes partes do mundo e diferentes países com estas regiões. Os clientes fazem diversos pedidos e cada pedido con-

tém muitas partes diferentes compradas de diferentes fornecedores a diferentes preços. A lista das partes e a lista de fornecedores são conectados por uma tabela para indicar as partes específicas fornecidas por cada fornecedor [Barata *et al.*, 2014]. Há também outro banco de dados analítico, carregado periodicamente com os dados, para uma estrutura de *Data Warehouse.*

O *benchmark* TPC-H não gera sua carga de teste em execução. A geração da carga de teste é feita externamente, por uma aplicação chamada DBGen[10]. Esta aplicação gera arquivos com os valores que serão inseridos como tuplas no schema do *benchmark*. A escala utilizada para geração de carga neste teste foi 1. Mesmo sendo apenas 1 unidade, o volume de registros já é suficiente para avaliar os aspectos necessários. Também foi setado o número de usuários virtuais que serão simulados para apenas 1.

4.3. Arquitetura física

Nesta seção é apresentado o ambiente em que rodaram os testes. Resumidamente, a arquitetura do teste consistiu de:

- 3 nodos físicos para hospedar o *cluster* que os produtos utilizaram. Os nodos continham igual configuração: Sistema Operacional Xubuntu 16.04 Server LTS 64 bits; Processador Intel Core Processador Intel® Core™ i5-7200, com 4 núcleos físicos de 2,50 GHz; Memória RAM 8GB DDR3 1333Mhz; Disco rígido de 320GB de velocidade 5400 RPM;
- 1 nodo físico para hospedar o aplicativo OLTP-Bench e enviar as requisições para os nodos do *cluster*. A configuração deste nodo contou com sistema operacional Ubuntu 18.04 Server LTS 64-bits; processador Intel Core Processador Intel® Core™ i7-6500U, com 4 núcleos físicos de *clock* real de 2,50 GHz; Disco de estado sólido de 250GB;
- Conexão *Ethernet* entre todas as instâncias apresentadas acima, com velocidade de 100 Mbps.

Utilizando o cluster local mencionado na seção anterior, foi realizada a instalação dos produtos para teste. As instalações buscaram seguir as configurações padrão dos produtos, não sendo feita nenhuma otimização, de modo a não favorecer nenhum dos resultados. Para os produtos CockroachDB, MemSQL e NuoDB foi realizada uma instalação local seguindo as orientações dos fabricantes para instalação[11][12][13]. Para o produto VoltDB foi utilizado a tecnologia de virtualização Docker, utilizando contêineres disponibilizados pelo fabricante[14].

4.4. Métricas avaliadas

Métricas são medidas quantitativas de modo a validar em várias dimensões o desempenho de sistemas. As métricas utilizadas neste trabalho foram:

- A análise da taxa de transações executadas no tempo (Throughput);

[10] https://github.com/electrum/tpch-dbgen
[11] https://www.memsql.com/install/
[12] https://www.cockroachlabs.com/docs/stable/install-cockroachdb.html
[13] https://www.nuodb.com/dev-center/community-edition-download
[14] https://hub.docker.com/r/voltdb/voltdb-community/

- Análise de latência das transações, através da análise da média geral das latências, análise dos percentis 90 e 99 e média das latências específicas de cada teste (por tipo de transação no YCSB e por consulta no TPC-H). O desvio padrão das amostras observadas também foram calculados para ajudar a explicar as observações

5. Análise dos resultados

Nesta seção são mostrados os resultados obtidos com base nos *benchmarks*, arquitetura e métricas definidos na seção anterior.

5.1. Resultados para o benchmark YCSB

A seguir observamos os resultados obtidos com o *benchmark* YCSB. O resultado foi coletado com dados de 5 execuções deste teste para cada produto. No Gráfico 1 analisamos a primeira métrica, do número de transações executadas por segundo ao decorrer do teste.

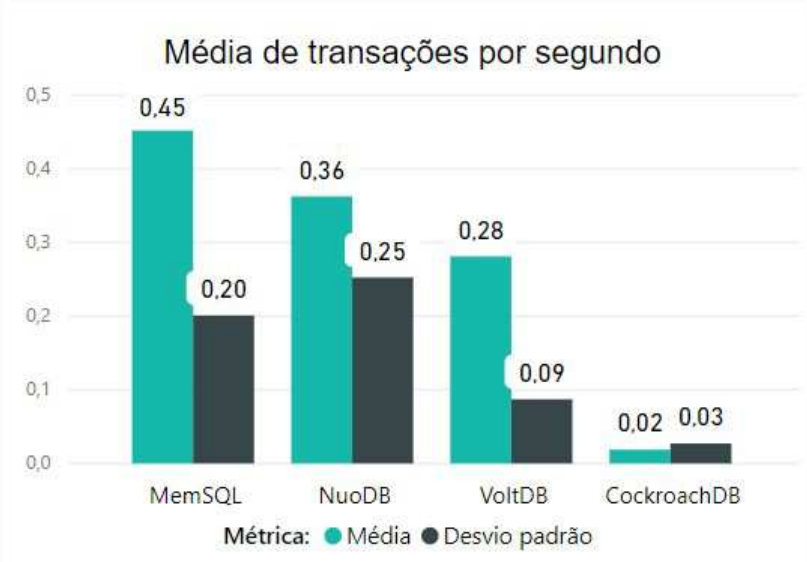


Figura 1. Médias de transações por segundo obtidas no *benchmark* YCSB.

Como podemos avaliar pela análise do gráfico, o MemSQL apresentou melhores resultados na categoria executando em média 0,45 transações a cada segundo do teste, a frente do NuoDB, que executou 0,36. VoltDB por sua vez executou 0,28 transações por segundo em média, e por último temos o Cockroach, executando 0,02 transações por segundo em média. O desvio padrão mostra que o grau de dispersão entre os resultados foi baixo, mostrando uma boa uniformidade na execução, sendo um puco mais discrepante na execução do MemSQL e NuoDB.

A diferença considerável obtida no Gráfico 1 do produto CockroachDB para os concorrentes pode ser explicada comparando as demais métricas. Outro fator importante para a compreensão do resultado é a média de latência das

transações executadas. O resultado de análise da latência média é mostrado no Gráfico 2.

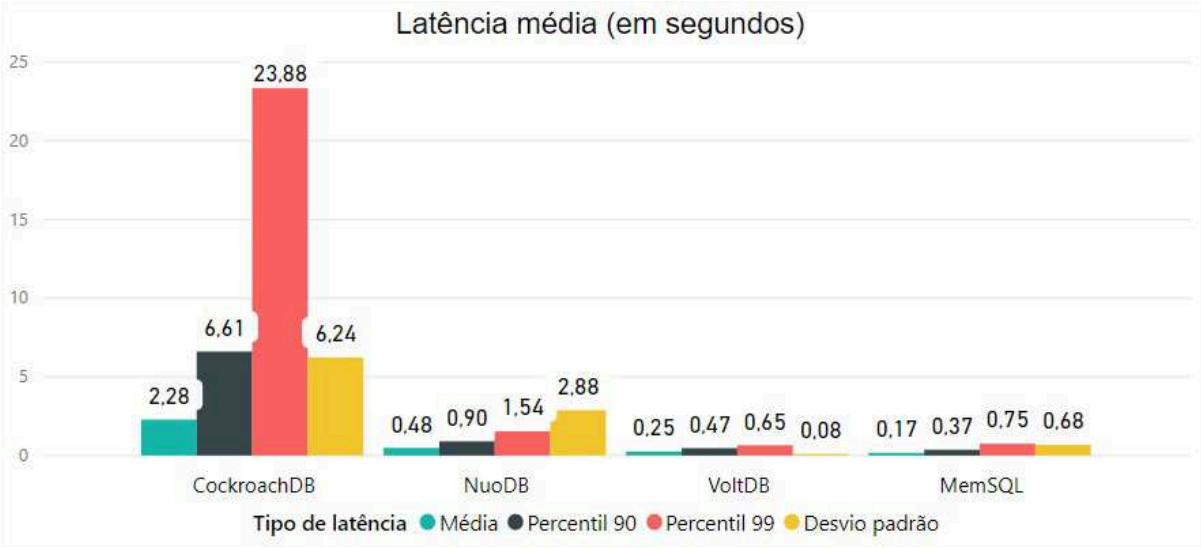


Figura 2. Latências médias obtidas no *benchmark* YCSB.

Como se pode observar no Gráfico 2, o CockroachDB também apresenta maior latência média nas transações, quando não se considera o tipo de transação. O desvio padrão mostra que o grau de dispersão entre os resultados foi alto nos casos do CockroachDB e NuoDB, mostrando que as observações dos resultados destes produtos tiveram uma grânde discrepância de valores, diferente dos outros produtos que mostraram uma execução mais uniforme do teste. Em relação a média, NuoDB e VoltDB sucedem o CockroachDB no *ranking*, mostrando resultados parecidos entre si. A melhor latência média fica por conta do MemSQL, com uma latência de 0,17 por segundo.

A análise dos percentis 90 e 99 mostra que quantidade expressiva das transações com maior latência se encontram em um pequeno número de transações situadas nesses pontos, representando mais que o dobro da média situada no percentil 99. O caso é mais evidente no uso do CockroachDB, onde temos uma grande discrepância entre a média de 2,28 segundos na média de latência, e 23,88 segundos na média de latências no percentil 99.

A análise da discrepância observada pelo produto CockroachDB em latência média e média de transações por segundo apresentadas deve considerar que o armazenamento principal do CockroachDB não é em memória principal (RAM) como os outros, e sim uma arquitetura *key-value* em disco. Este banco de dados utiliza um percentual da memória principal (RAM) para cache, mas utiliza grande parte de suas operações baseadas em disco, o que gerou as latências e taxa de vazão observados.

As discrepâncias também são notadas no produto NuoDB. Tanto na média de transações por segundo, quanto na análise de latência média, o produto mostra um grau de dispersão um pouco elevado entre os resultados. Tais resultados

tendem a ocorrer tendo em vista as limitações da versão gratuita, onde é possível utilizar o armazenamento em apenas um dos nodos do *cluster*, mesmo que seja possível utilizar o componente de execução de transações em três nodos. O armazenamento no *storage* único pode aumentar a latência da execução, visto que os nodos executam parte do teste em um nodo, transferindo todo o resultado para o nodo que salva os dados.

Por sua vez, o VoltDB ocupa o terceiro lugar no teste. O VoltDB, produto escrito em Java, pode em sua execução ter influência do *garbage colector* da própria Java Virtual Machine (JVM), o que deve ser um ponto a considerar nos resultados dos *benchmarks* avaliados neste trabalho. Mesmo assim, obteve boas médias nas métricas observadas.

Ao se considerar as médias de latência não comparamos qual a transação o banco de dados está fazendo. Avaliando o tipo de transação, podemos detectar se um dos candidatos tem melhor desempenho em inserts ou selects, por exemplo. O YCSB fornece como saída informações de latência que discriminam o tipo de transação realizado, permitindo essa análise.O Gráfico 3 mostra a média de latência, agora considerando o tipo de transação que o banco de dados está executando.

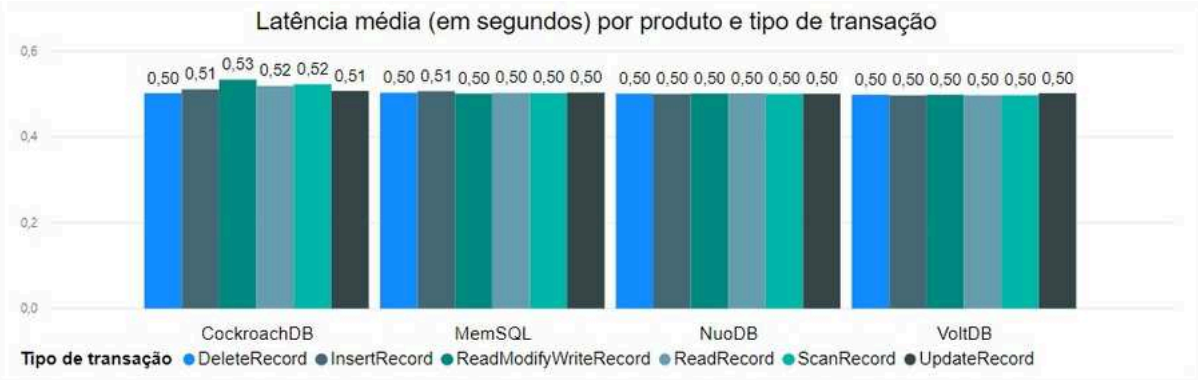


Figura 3. Latências médias do teste discriminadas por tipo de transação no *benchmark* YCSB.

Analisando o Gráfico 3, pode-se inferir que não há diferença significante entre o tipo de execução executada. As transações utilizadas para medir estes parametros são muito pequenas, assim quando o BD alvo recebe é capaz de executa-la imediatamente. Desta forma, os tempos apresentados ficam limitados a latência de rede, o que resulta nas latências observadas estarem tão próximas. A diferença fica na ordem de microssegundos. Apenas o CockroachDB oscila um pouco nas latências, se mostrando melhor executando a carga de deleção de registros e pior na inserção. Os demais mantêm uma média de 0,50 segundo de latência para todos os tipos de transação.

5.2. Resultados para o benchmark Voter

Na avaliaçãoavaliação do *benchmark* Voter, utilizamos um tempo limitado a 300 segundos, utilizando um fator de escala 10. O resultado foi coletado

com dados de 5 execuções deste teste para cada produto. No gráfico abaixo, analisamos a primeira métrica, do número de transações executadas por segundo ao decorrer do teste.

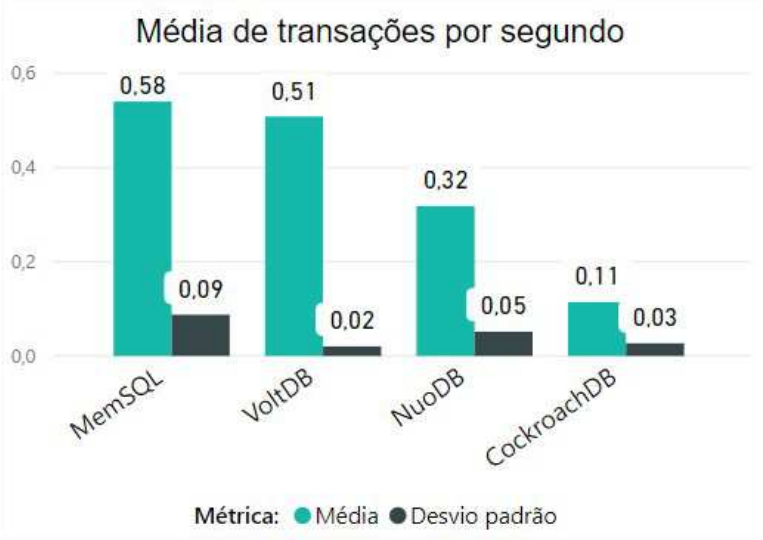


Figura 4. Médias de transações por segundo obtidas no *benchmark* Voter.

Analisando o Gráfico 4, novamente temos o MemSQL com melhores resultados na categoria, executando em média 0,58 transações a cada segundo do teste, seguido do VoltDB, que executou 0,51. NuoDB ficou intermediário entre o segundo e o último lugar, novamente ocupado pelo CockroachDB. O desvio padrão mostra que o grau de dispersão entre os resultados foi muito baixo, mostrando uma boa uniformidade na execução. Logo após, analisamos a latência média obtida pelos produtos. O gráfico abaixo mostra os resultados obtidos pelos concorrentes:

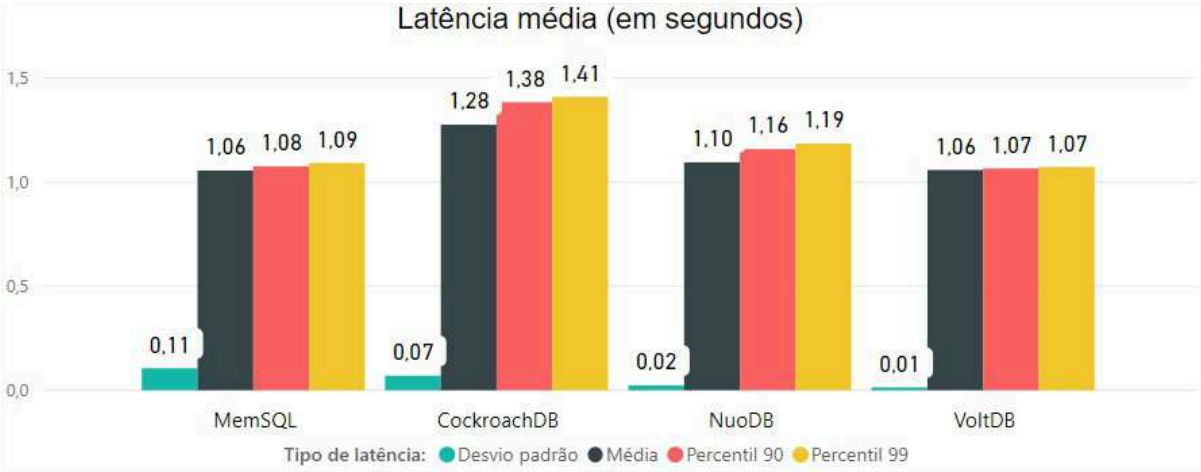


Figura 5. Latência média total nas transações obtidas para cada produto no *benchmark* Voter.

Observamos novamente uma maior latência obtida pelo produto CockroachDB. O desvio padrão mostra que o grau de dispersão entre os resultados foi muito baixo, mostrando uma boa uniformidade na execução. No teste do Voter temos uma situação muito diferente da observada no teste de latência média geral com YCSB no Gráfico 2. No Voter a variação obtida entre um produto e outro é pequena, tanto nas análises médias quanto nas de percentil 90 e 99, o que sugere uma distribuição uniforme entre as latências no teste. Isso demonstra que em transações menores, como é o caso deste benchmark, os produtos operam de forma semelhante.

O Voter visa uma exploração mais acentuada das características que se espera de um banco NewSQL. Como apresentado anteriormente, suas transações consistem em uma votação por número de telefone em um participante do "American Idol". Assim, este *benchmark* acarreta um estresse maior sobre os produtos testados, simulando diversos usuários votando em seu participante favorito. Os BDs MemSQL e VoltDB apresentaram resultados muito semelhantes, demostrando que são capazes de lidar com várias requisições simultâneas com baixa latência. O NuoDB apresenta uma arquitetura com níveis mais complexos de armazenamento, o que acaba prejudicando um pouco seu desempenho com um grande volume de transações mais simples. Já o Cockroach apresentou o pior resultado. Sua arquitetura, apesar de trazer novos algoritmos ainda fica limitada a algumas operações que utilizam o disco, o que deprecia muito seu desempenho em comparação aos demais.

5.3. Resultados para o benchmark TPC-H

O TPCH é um *benchmark* consolidado na avaliação de BDs com arquitetura distribuída na nuvem. Ele é o mais robusto dos três *benchmarks* analisados e possui em sua composição consultas OLTP e OLAP. Seu maior foco é em consultas mais complexas envolvendo contagens e somas para relatórios gerenciais (processamento OLAP). Apesar de sistemas NewSQL terem maior foco em OLTP, neste trabalhamos consideramos este *benchmark* para avaliar se as soluções propostas são flexíveis o suficiente para se adaptarem a cargas OLAP. O resultado foi coletado com dados de 3 execuções deste teste para cada produto.

Apenas foi possível obter os resultados de execução para o BD MemSQL. Os resultados apresentados nos gráficos a seguir demonstram que este BD não é totalmente preparado para este tipo de cargas. O Gráfico 6 demonstra que o BD foi capaz de tratar 1,4 transações por segundo. O desvio padrão mostra que o grau de dispersão entre os resultados foi médio. Porém o Gráfico 6 demonstra que a latência foi muito superior do que a observada nos experimentos com os demais benchmarks. Isso se deve ao fato das consultas apresentarem um nível de complexidade maior, apresentado somas e contabilizações sobre os dados.

No gráfico abaixo analisamos a primeira métrica, do número de transações executadas por segundo ao decorrer do teste.

Figura 6. Média de transações por segundo obtida no *benchmark* TPC-H.

O resultado de análise da latência média é mostrado no gráfico abaixo.

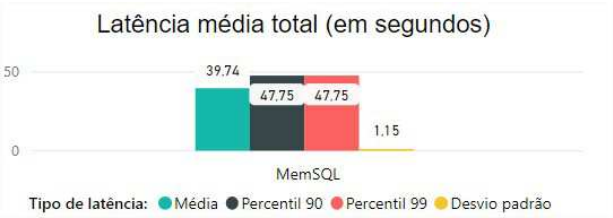


Figura 7. Latência média total nas transações obtidas para o *benchmark* TPC-H.

O Gráfico 8 apresenta a latência para cada uma das 22 consultas aplicadas pelo *benchmark*. O desvio padrão mostra que o grau de dispersão entre os resultados foi médio. Consultas como Q2, Q6, Q7,Q8 e Q19 apresentam seleções mais simples e mais restritivas, com baixa complexidade, permitindo uma latência menor. Já consultas como Q9 e Q18 são consultas mais complexas e apresentam filtros menos restritivos aplicando as suas operações em um conjunto maior de dados.

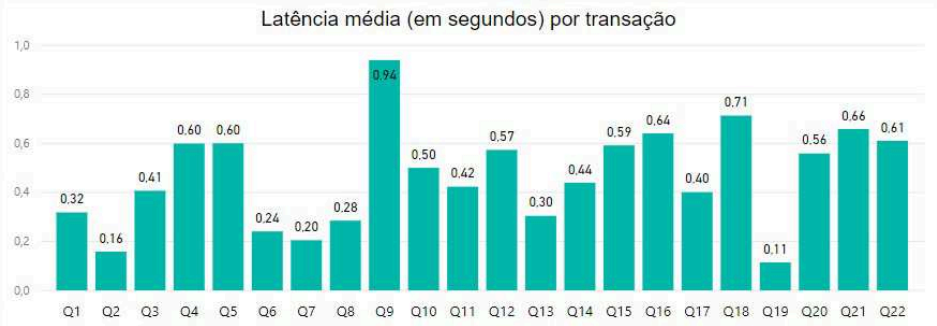


Figura 8. Latência média do teste, discriminada por transação executada, obtida no *benchmark* TPC-H.

Como mencionado anteriormente, foi possível executar este *benchmark* apenas para o MemSQL o que demonstra uma alta flexibilidade neste sistema.

Nas tentativas de execução realizadas para o VoltDB esbarrou-se em uma limitação técnica imposta pelo próprio fornecedor. Como sua arquitetura é otimizada para execução de transações OLTP, o sistema não permite que uma tabela resultado ultrapase 1 GB (100 MB é o valor padrão), o que acarreta em erro e aborto da transação. O NuoDB realizou a importação dos dados rapidamente, porém travava na execução das consultas sem apresentar nenhum tipo de erro nos logs principais. O Cockroach não conseguiu realizar a carga dos dados, mesmo possuindo memória RAM suficiente para importar os 6GB do banco de dados do TPC-H, apresentou consumo muito elevado de memória esgotando os 8GB disponíveis no nodo para fazer esse carregamento, travando o nodo.

6. Conclusões e comentários finais

O presente trabalho analisou quatro produtos que se baseiam no paradigma NewSQL. Abordou-se o contexto acerca do paradigma NewSQL e foram explanadas características pertinentes de cada um dos produtos. Realizou-se a instalação de um ambiente distribuído em *cluster* físico com 3 nodos, de forma de comparar o comportamento das transações, nas nuances de cada uma das arquiteturas. Foi empregada a técnica de *benchmark*, utilizando de benchmarks de domínio específico, através de um *framework* chamado OLTP-Bench.

Sobre os resultados obtidos das análises verificou-se que geralmente o produto MemSQL se manteve a frente nas características observadas, obtendo alta taxa de *throughput*, e baixa latência nos contextos analisados. Também foi o único produto que conseguiu executar as cargas do teste TPCH mostrando-se mais flexível que os demais produtos. Os produtos VoltDB e NuoDB se comportaram de maneira semelhante na maioria dos resultados analisados, mesmo com as considerações sobre as restrições da versão grátis utilizada pelo NuoDB e uma possível influência da linguagem de construção do VoltDB. Mostraram no geral uma boa execução das cargas analisadas, porém não conseguiram rodar o TPC-H para fornecer resultados sobre suas capacidades OLAP.

O banco de dados CockroachDB mostrou os piores resultados, com discrepâncias muito consideráveis nos contextos observados, principalmente nas taxas médias de transações por segundo. Os resultados com o CockroachDB se mostram justificáveis quando observado o fato que o componente interno RocksDB e o cache interno do CockroachDB alternam o uso da memória RAM e o disco para armazenamento e execução das transações, diferentemente dos outros produtos que utilizam a memória RAM como memória principal para todas as operações. O banco de dados também não completou a execução dos testes TPC-H, afetando a análise OLAP do produto.

Os bancos de dados analisados mostram também que produtos NewSQL são uma alternativa interessante aos bancos de dados NoSQL, conseguindo entregar boas métricas de performance, mantendo as características ACID e resolvendo assim o impasse do teorema CAP de forma satisfatória. A redução natural no preço das tecnologias, como memória RAM, com o tempo tornarão esses produtos ainda mais atrativos, incentivando seu uso a problemas que utilizavam do paradigma NoSQL como alternativa.

Em vista dos resultados e conclusões obtidos com este trabalho, mesmo que ao encontro dos objetivos propostos, há possibilidades de complementá-lo, levantando então as seguintes sugestões de trabalhos futuros:

- Utilização de um benchmark adicional que possa avaliar o comportamento dos produtos para as capacidades no processamento OLAP dos bancos de dados observados;
- Realizar uma análise semelhante utilizando particionamento geográfico dos nodos para uma verificação mais aprofundada das características analisadas;
- Complementar o resultado com a criação de métricas auxiliares e monitoramento de outras dimensões, como, por exemplo, a utilização dos recursos de hardware pelo sistema operacional, e a criação de threads na execução, para agregar o entendimento sobre os resultados obtidos.
- Considerar a avaliação de bancos de dados de outros paradigmas envolvendo as mesmas métricas, para afirmar e comprovar as vantagens que o paradigma NewSQL propõe frente aos paradigmas relacional e/ou NoSQL.

Referências

Barata, M., Bernardino, J., e Furtado, P. (2014). YCSB and TPC-H: Big data and decision support benchmarks. *2014 IEEE International Congress on Big Data*, pages 800 – 801.

Difallah, D. E., Pavlo, A., Curino, C., e Cudre-Mauroux, P. (2013). OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *Proceedings ofthe VLDB Endowment*, **7**(4), 277 – 288.

Grolinger, K., Higashino, W. A., Tiwari, A., e Capretz, M. A. (2013). Data management in cloud environments: NoSQL and NewSQL data stores. *Journal of Cloud Computing: Advances, Systems and Applications*, **2**(1).

Gurevich, Y. (2015). Comparative Survey of NoSQL/ NewSQL DB Systems. (December).

Jelly, I., Kerridge, J., e Bates, C. (1994). Benchmarking parallel SQL database machines. **826**, 105 – 120.

Kaur, K. e Sachdeva, M. (2017). Performance Evaluation of NewSQL Databases. *International Conference on Inventive Systems and Control*, pages 1 – 5.

Labs, C. (2018). Architecture Overview.

MemSQL (2018). MemSQL Architecture: Technology Innovations Power Convergence of Transactions and Analytics.

Pavlo, A. (2016). What's Really New with NewSQL? *SIGMOD Record*, **45**(2).

VoltDB (2013). VoltDB Technical Overview.

VoltDB, Inc (2018). Using VoltDB.